

Distributed and Parallel Databases, 18, 9–36, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Enhancing ebXML Registries to Make them OWL Aware*

ASUMAN DOGAC asuman@srdc.metu.edu.tr
YILDIRAY KABAK yildiray@srdc.metu.edu.tr
GOKCE B. LALECI banu@srdc.metu.edu.tr
Software Research and Development Center, Department of Computer Engineering, Middle East Technical University (METU), 06531, Ankara, Turkiye

CARL MATTOCKS carlmattocks@checkmi.com
CHECKMi, 225 Emerson Lane, Berkeley Heights, NJ 07922, USA

FARRUKH NAJMI farrukh.najmi@sun.com
Sun Microsystems, Inc., 1 Network Dr., Burlington, MA 01810, USA

JEFF POLLOCK jeff.pollock@networkinference.com
Cerebra, Inc., 5963 La Place ct., Carlsload, CA 92008, USA

Recommended by: Athman Bouguettaya and Boualem Benatallah

Abstract. ebXML is a standard from OASIS and UN/CEFACT which specifies an infrastructure to facilitate electronic business. In this paper, we address how ebXML registry semantics support can be further enhanced to make it OWL aware. OWL constructs are represented through ebXML registry information model constructs, and stored procedures are defined in the ebXML registry for processing the OWL semantics. These predefined stored queries provide the necessary means to exploit the enhanced semantics stored in the registry. In this way, an application program does not have to be aware of the details of how this semantics support is achieved in ebXML registry, and does not have to contain additional code to process this semantics.

We believe that this approach is quite powerful to associate semantics with registry objects: it becomes possible to retrieve knowledge through queries, the enhancements to the registry are generic and also the registry specification is kept intact. The capabilities provided move the semantics support beyond what is currently available in ebXML registries and it does so by using a standard ontology language.

To be able to demonstrate the benefits of the enhancements, we also show how the resulting semantics can be made use of in Web service discovery and composition.

Keywords: semantics, ebXML registry, Web Ontology Language (OWL), semantic web service discovery

*This work is supported in part by the European Commission, Project No: IST-1-002104-STP SATINE and the Scientific and Technical Research Council of Turkey (TÜBİTAK), Project No: EEEAG 104E013 and is realized as a proposal to OASIS ebXML Semantic Content Management subcommittee which is working on possible semantic extensions to the registry.

1. Introduction

Electronic Business XML (ebXML) [14] is a standard from OASIS [27] and United Nations Centre for Trade Facilitation and Electronic Business, UN/CEFACT [45]. ebXML provides an infrastructure that allows enterprises to find each other's services, products, business processes, and documents in a standard way and thus helps to facilitate conducting electronic business. One of the important characteristics of ebXML compliant registries is that they provide mechanisms to define and associate semantics with registry objects. The ebXML "Registry" component holds the metadata for the registry objects and the documents pointed at by the registry objects reside in an ebXML repository. The basic semantic mechanisms of ebXML Registry are classification hierarchies consisting of classification nodes and the predefined associations among classification nodes. These are all registry objects and registry objects can be assigned properties through a slot mechanism. Given these constructs, considerable amount of semantics can be defined in the registry.

However, currently semantics is becoming a much broader issue than it used to be since several application domains are making use of ontologies to add the knowledge dimension to their data and applications [9, 17, 42]. One of the driving forces for ontologies is the Semantic Web initiative [2]. As a part of this initiative, W3C's Web Ontology Working Group defined Web Ontology Language (OWL) [31]. Naturally, there is lot to be gained from using a standard ontology definition language, like OWL, to express semantics in ebXML registries.

In this paper, we investigate how ebXML registries can be made OWL aware. There are three alternatives to support OWL ontologies through ebXML registries (Table 1):

Table 1. Three approaches to support OWL Ontologies through ebXML registries.

Method	Advantages	Disadvantages
–Representing OWL semantic constructs through ebXML Registry constructs	–No changes in the registry architecture specification and implementation –OWL semantics stored in an ebXML registry can be retrieved from the registry through native ebXML query facilities written by the user	–Further processiong needs to be done by the application program to make use of the enhanced semantics
–Providing predefined procedures to process OWL semantics in ebXML registry	–The user can call stored procedures when the need arises –The stored procedures can also be called transparently to the user by changing only the query manager component of the registry	–Limited reasoning capabilities
–Changing ebXML registry specification to support OWL with full reasoning capabilities	–Supports OWL with full reasoning capabilities –Makes it possible to deduce new data that is not directly stored in the registry	–Requires considerable changes in the registry architecture –Brings about the efficiency considerations of rule based systems

- Various constructs of OWL can be represented by ebXML classification hierarchies with no changes in the registry architecture specification and implementation. In this way, although some of the OWL semantics stored in an ebXML registry can be retrieved from the registry through ebXML query facilities, further processing needs to be done by the application program to make use of the enhanced semantics. For example, we can introduce "subClassOf" "association" to the ebXML registry to handle OWL multiple inheritance. Yet since ebXML registry does not natively support such an association type, to make any use of this semantics, the application program must have the necessary code, say, to find out all the super classes of a given class.
- To improve on the first alternative, the code to process the OWL semantics can be defined through generic stored procedures and be made available from the ebXML registry. For example, to find the super classes of a given class (defined through a new association type of "subClassOf"), a generic stored procedure can be defined. The user can call this procedure when the need arises. Furthermore, the stored procedures can also be called transparently to the user by the query manager. This involves an update in the query manager component of the registry. We believe that this approach is quite powerful to associate semantics with registry objects: it becomes possible to retrieve knowledge through queries and the enhancements to the registry are generic. Hence we take this approach.
- The third approach is changing the ebXML registry architecture to support OWL with full reasoning capabilities. Reasoning entails the derivation of new instances that are not directly stored in the registry. To deduce this data, rules need to be stored in the registry. However, this approach requires considerable changes in the registry architecture and brings about the efficiency considerations of rule based systems. Since our aim is to make ebXML registry OWL aware rather than specifying a new registry architecture, this approach will not be pursued any further in this paper.

There is another decision to be made: OWL is defined as three different sublanguages: *OWL Full*, *OWL DL* and *OWL Lite*, each geared towards fulfilling different requirements [25]. We choose *OWL Lite* since ebXML registries are for industrial use. Currently, the industry is using taxonomies like Universal Standard Products and Services Classification (UNSPSC) [46] and North American Industrial Classification Scheme (NAICS) codes for semantic descriptions and *OWL Lite* provides a quick migration path for taxonomies [41]. For *OWL Full*, it is unlikely that any reasoning software will be available [25]. *OWL DL* constructs, on the other hand, can be exploited best through the reasoners and as we have previously mentioned, using a reasoner natively in the registry requires changes in the ebXML registry architecture specification and this is not our purpose. Yet, for those who wish to represent *OWL Full* or *OWL DL* constructs, the mechanisms we present for OWL Lite can easily be adopted.

For ebXML registries, being OWL aware entails the following:

- Representing OWL constructs through ebXML Registry Information Model (RIM) constructs: For this purpose we show how OWL constructs can be expressed through ebXML registry semantic constructs.
- Automatically generating ebXML constructs from the OWL descriptions and storing the resulting constructs into the ebXML registry: We developed a tool to create an ebXML

Classification Hierarchy from a given OWL ontology automatically by using the transformations described in Section 4.

- Facilitating the querying of the registry for enhanced semantics: ebXML allows to annotate registry objects with classification nodes and ebXML query facilities allow these explicit relationships to be queried. However, when we introduce OWL semantics to the registry, there is a need for application code to do the necessary processing. For example, we may define a number of classes to be equivalent and we may wish to retrieve the instances of all the equivalent classes in the registry. In ebXML, to do this, a user must issue a number of seperate queries, using the result of a previous query as an input to the next query. To overcome this burden on the user side, we have provided a number of generic stored procedures to be invoked by the user when necessary. Note that these procedures can be invoked transparently to the user by making the necessary changes in the query manager component of the ebXML registry.

Furthermore, we show how the resulting semantics can be made use of in Web service discovery and composition.

This work is realized within the scope of IST-2104 SATINE project [23] as a proposal to OASIS ebXML Semantic Content Management subcommittee which is working on possible semantic extensions to the registry.

The paper is organized as follows: Section 2 briefly summarizes the main technologies involved in this work, namely, OWL and ebXML Registry architecture. In Section 3, we give an overall view of the approach and describe how the proposed enhancements fit into ebXML architecture. Section 4 describes how semantics defined in OWL ontologies can be represented and accessed in ebXML registries. Section 5 gives the related work. In this secion, we also provide a summary of the ebXML semantic standardization efforts undertaken by the OASIS open source standards body. Finally, Section 6 concludes the paper and presents the future work.

2. OWL and ebXML RIM

In order to describe how OWL ontologies can be stored in ebXML registries we first briefly summarize the semantic constructs they each provide.

2.1. Web Ontology Language (OWL)

Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies on the World Wide Web [31]. OWL is derived from the DAML+OIL Web Ontology Language [1, 7] and builds upon the Resource Description Framework (RDF) [37, 38].

OWL describes the *structure* of a domain in terms of *classes* and *properties*. Classes can be names (URIs) or *expressions* and the following set of *constructors* are provided for building class expressions: *owl:intersectionOf*, *owl:unionOf*, *owl:complementOf*, *owl:oneOf*, *owl:allValuesFrom*, *owl:someValuesFrom*, and *owl:hasValue*.

RDF Schema Features	**(In)Equality**	**Property Characteristics**
– *Class (Thing, Nothing)*	– *equivalentClass*	– *ObjectProperty*
– *rdfs: subClassOf*	– *equivalentProperty*	– *DatatypeProperty*
– *rdf:Property*	– *sameAs*	– *inverseOf*
– *rdfs: subPropertyOf*	– *differentFrom*	– *TransitiveProperty*
– *rdfs:domain*	– *AllDifferent*	– *SymmetricProperty*
– *rdfs:range*	– *distinctMembers*	– *FunctionalProperty*
– *individual*		– *InverseFunctionalProperty*
Property Restrictions	**Restricted Cardinality**	**Datatypes**
– *Restriction*	– *minCardinality*	–*xsd datatypes*
– *onProperty*	– *maxCardinality*	**Class Intersection**
– *allValuesFrom*	– *cardinality*	–*intersectionOf*
– *someValuesFrom*		

Figure 1. OWL lite constructs.

In OWL, properties can have multiple domains and multiple ranges. Multiple domain (range) expressions restrict the domain (range) of a property to the intersection of the class expressions.

Another aspect of the language is the axioms supported. These axioms make it possible to assert subsumption or equivalence with respect to classes or properties [19]. The following are the set of OWL axioms: *rdfs:subClassOf*, *owl:sameClassAs*, *rdfs:subPropertyOf*, *owl:samePropertyAs*, *owl:disjointWith*, *owl:sameIndividualAs*, *owl:differentIndividualFrom*, *owl:inverseOf*, *owl:transitiveProperty*, *owl:functionalProperty*, and *owl:inverseFunctionalProperty*. OWL constructs are given in more detail in Section 4.1 while describing their representation in ebXML registry.

OWL provides three decreasingly expressive sublanguages [41]:

- *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. It is unlikely that any reasoning software will be able to support complete reasoning for *OWL Full* [25].
- *OWL DL* supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). *OWL DL* is so named due to its correspondence with description logics which form the formal foundation of OWL.
- *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints.

Within the scope of this paper, we consider *OWL Lite* constructs which are given in figure 1 and in the rest of the paper, OWL is used to mean *OWL Lite* unless otherwise stated.

2.2. ebXML registry architecture and information model

An ebXML registry consists of both a registry and a repository. The repository is capable of storing any type of electronic content, while the registry is capable of storing metadata that describes content. The content within the repository is referred to as “repository items” while the metadata within the registry is referred to as “registry objects”. Clients access the

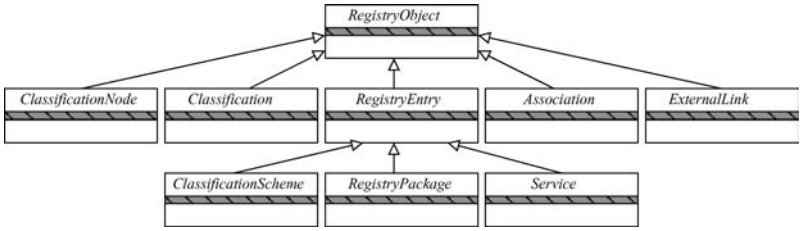


Figure 2. A part of the ebXML RIM class hierarchy.

registry and the repository via the ebXML registry API as defined in [16]. The API has two main interfaces:

- LifeCycleManager (LCM) is the interface responsible for all object lifecycle management requests.
- QueryManager (QM) is the interface responsible for handling all query requests.

The LifeCycleManager service enforces the life cycle rules for objects. The QueryManager interface of the ebXML Registry API provides access to the query service of the ebXML registry. A client uses the operations defined by this service to query the registry and discover objects. Supported query syntaxes include:

- An XML Filter Query syntax,
- An SQL-92 query, and
- A stored query syntax that allows client to invoke queries stored in the server by simply identifying the parameterized query and providing parameters for the query.

2.2.1. ebXML registry information model. The ebXML registry defines a Registry Information Model (RIM) [15] which specifies the standard metadata that may be submitted to the registry. This complements the ebXML Registry API which defines the interface clients may use to interact with the registry. Figure 2 presents the part of the ebXML RIM [15] related with storing metadata information. The main features of the information model include:

- *Registry object*: The top level class in RIM is the "RegistryObject". This is an abstract base class used by most classes in the model. It provides minimal metadata for registry objects.
- *Object identification*: All RegistryObjects have a globally unique id, a human friendly name and a human friendly description.
- *Slot*: "Slot" instances provide a dynamic way to add arbitrary attributes to "RegistryObject" instances.
- *Object classification*: Any RegistryObject may be classified using ClassificationSchemes and ClassificationNodes which represent individual class hierarchy elements. A

Table 2. Predefined association types in ebXML registries.

Name	Description
Related To	Defines that source RegistryObject is related to target RegistryObject.
HasMember	Defines that the source RegistryPackage object has the target RegistryObject object as a member.
ExternallyLinks	Defines that the source ExternalLink object externally links the target RegistryObject object.
Contains	Defines that source RegistryObject contains the target RegistryObject.
EquivalentTo	Defines that source RegistryObject is equivalent to the target RegistryObject.
Extends	Defines that source RegistryObject inherits from or specializes the target RegistryObject.
Implements	Defines that source RegistryObject implements the functionality defined by the target RegistryObject.
InstanceOf	Defines that source RegistryObject is an Instance of target RegistryObject.
Supersedes	Defines that the source RegistryObject supersedes the target RegistryObject.
Uses	Defines that the source RegistryObject uses the target RegistryObject in some manner.
Replaces	Defines that the source RegistryObject replaces the target RegistryObject in some manner.
SubmitterOf	Defines that the source Organization is the submitter of the target RegistryObject.
ResponsibleFor	Defines that the source Organization is responsible for the ongoing maintainence of the target RegistryObject.
OffersService	Defines that the source Organization object offers the target Service object as a service.

ClassificationScheme defines a tree structure made up of "ClassificationNodes". The ClassificationSchemes may be user-defined.

- *Object association*: Any RegistryObject may be associated with any other RegistryObject using an Association instance where one object is the sourceObject and the other is the targetObject of the Association instance. An Association instance may have an associationType which defines the nature of the association. There are a number of predefined *Association Types* that a registry must support to be ebXML compliant [15] as shown in Table 2. ebXML allows this list to be expanded.
- *Object organization*: RegistryObjects may be organized in a hierarchical structure using a familiar file and folder metaphor. The RegistryPackage instances serve as folders while RegistryObjects serve as files in this metaphor. In other words RegistryPackage instances group logically related RegistryObject instances together.
- *Service description*: The Service, ServiceBinding and SpecificationLink classes provide the ability to define service descriptions including WSDL and ebXML CPP/A.

As a summary, ebXML registry provides a persistent store for registry content. The current registry implementations store registry data in a relational database. ebXML Registry Services Specification defines a set of Registry Service interfaces which provide access to registry content. There are a set of methods that must be supported by each interface. A registry client program utilizes the services of the registry by invoking methods on one of

these interfaces. The Query Manager component also uses these methods to construct the objects by obtaining the required data from the relational database through SQL queries. In other words, when a client submits a request to the registry, registry objects are constructed by retrieving the related information from the database through SQL queries and are served to the user through the methods of these objects.

3. Proposed enhancements to the ebXML registry architecture

Being OWL aware entails the following enhancements to the ebXML registry:

- *Representing OWL constructs through ebXML constructs*: ebXML provides a classification hierarchy made up of classification nodes and predefined type of associations between the registry objects. We represent *OWL Lite* constructs by using combinations of these constructs and define additional types of associations when necessary. For example, "OWL ObjectProperty" is defined by introducing a new association of type "objectProperty". The details of this work are presented in Section 4.
- *Automatically generating ebXML constructs from the OWL descriptions and storing the resulting constructs into the ebXML registry*: We developed a tool to create an ebXML Classification Hierarchy from a given OWL ontology automatically by using the transformations described in Section 4. The OWL file is parsed using Jena [24], the classes together with their property and restrictions are identified, and the "SubmitObjectsRequest" is prepared automatically. This request is then sent to ebXML registry which in turn creates necessary classes and associations between them.
- *Querying the registry for enhanced semantics*: We provide additional stored procedures to process the OWL semantics introduced in Section 4. A user can handle the OWL semantics by using these stored procedures or through SQL. Note that stored procedures and SQL are two of the supported query syntaxes in ebXML. In order to handle the OWL semantics transparently to the user through the third query syntax, namely, the "filter query", the Query Manager needs to be modified to invoke the related stored procedures we have introduced, when necessary.

The enhanced architecture is shown in figure 3. The OWL constructs are represented entirely through ebXML constructs by defining new types of associations which is allowed by the registry architecture. Hence there are no changes in the relational database schemas.

4. Providing OWL support to ebXML registries

In this section, we first describe how OWL constructs can be represented through ebXML registry information model constructs. We then provide the stored procedures to retrieve richer sets of results from the registry based on *OWL Lite* constructs. The stored procedures are defined using the ebXML relational schema specifications. The schemas used in the examples are given in Table 3.

Table 3. ebXML relational schemas.

ClassificationNode(accessControlPolicy, id, objectType, code, parent, path)
Association(accessControlPolicy, id, objectType, associationType, sourceObject, targetObject, isConfirmedBySourceOwner, isConfirmedByTargetOwner)
Name_(charset, lang, value, parent)

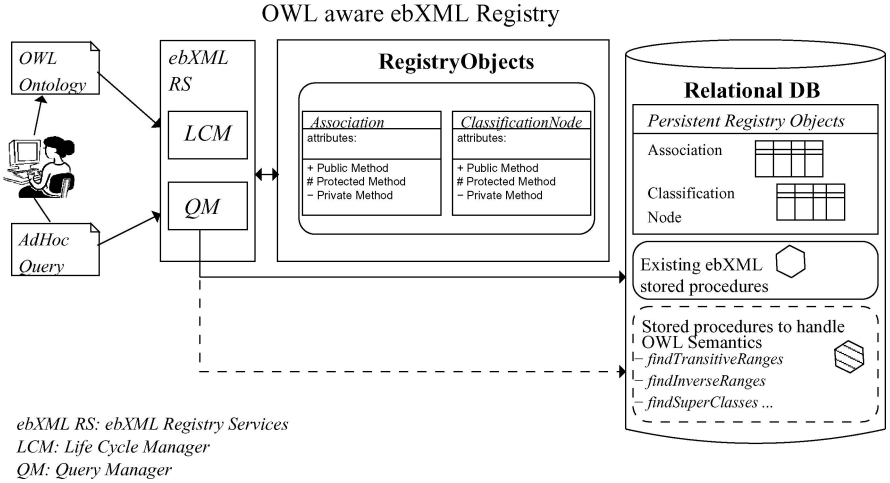


Figure 3. Enhancements to the ebXML registry architecture.

In Section 4.2, to demonstrate the benefits of the additional semantics incorporated into the ebXML registries, we describe a semantic-based service composition tool. This tool partially automates service discovery and composition in OWL aware ebXML registries.

4.1. Mapping OWL ontologies through ebXML classification hierarchies and providing registry support for processing the OWL constructs

From the descriptions presented in Section 2, it is clear that there are considerable differences between an OWL ontology and an ebXML class hierarchy in terms of semantic constructs. In this section, we provide the details of representing the *OWL Lite* constructs in an ebXML registry and then give the required stored procedures to process this semantics. Note that the mechanisms to represent currently available OWL Lite constructs can be thought of as a model for handling new OWL constructs that may appear or changes in existing constructs.

4.1.1. OWL classes and properties. OWL classes can be represented through "ClassificationNodes" and RDF properties that are used in OWL, can be treated as "Associations". An "Association" instance represents an association between a "source RegistryObject" and a "target RegistryObject". Hence the target object of "rdfs:domain" property can be mapped

to a "source RegistryObject" and the target object of "rdfs:range" can be mapped to a "target RegistryObject". In OWL, properties can be of two types:

- ObjectProperty type defines relations between instances of two classes.
- DatatypeProperty type defines relations between instances of classes and XML Schema datatypes.

To represent OWL ObjectProperty (or DatatypeProperty) in ebXML, we define a new type of association called "ObjectProperty" (or "DatatypeProperty"). Consider the following example which defines an object property "hasAirport" whose domain is "City" and whose range is "Airport":

```
<owl:ObjectProperty rdf:ID="hasAirport">
   <rdfs:domain rdf:resource="#City"/>
   <rdfs:range rdf:resource="#AirPort"/>
</owl:ObjectProperty>
```

In order to define this property in ebXML RIM, first, two classification nodes are created, namely "City" and "Airport". Then, an association, called "hasAirport" of type "ObjectProperty", is defined where the "sourceObject" is "City" and the "targetObject" is "Airport", as shown in the following:

```
<rim:ClassificationNode id='City' parent='Country'>
  </rim:ClassificationNode>
<rim:ClassificationNode id='Airport' parent='TravelThing'>
  </rim:ClassificationNode>
<rim:Association id='hasAirport' associationType='ObjectProperty'
  sourceObject = 'City' targetObject='Airport' >
</rim:Association>
```

Similarly, to represent OWL DatatypeProperty in ebXML, we define a new type of association called "DatatypeProperty". Consider the following example which defines an datatype property "hasPrice" whose domain is the "AirReservationServices" and whose range is "XMLSchema nonNegativeInteger":

```
<owl:DatatypeProperty rdf:ID="hasPrice">
  <rdfs:subpropertyOf rdf:resource="http://www.daml.org/services/
     daml-s/2001/05/Profile.owl"/>
  <rdfs:domain rdf:resource="#AirReservationServices"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema/
     nonNegativeInteger"/>
</owl:DatatypeProperty>
```

To describe this semantics, we define a new association of type "DatatypeProperty" as shown in the following:

```
<rim:Association id = 'hasPrice' associationType = 'DatatypeProperty'
   sourceObject = 'AirReservationServices'
```

```
   targetObject = 'integer' >
   <rim:Name> <rim:LocalizedString value ="hasPrice"/></rim:Name>
</rim:Association>
```

OWL allows the use of XML Schema datatypes to describe part of the datatype domain by simply including their URIs within an OWL ontology. In ebXML, XML Schema datatypes are used by providing an external link from the registry, as demonstrated in the following:

```
<rim:ExternalLink id = "integer"
              externalURI="http://www.w3.org/2001/XMLSchema#integer">
      <rim:Name> <rim:LocalizedString value = "XML Schema integer"/>
                 </rim:Name>
</rim:ExternalLink>
```

Once such ObjectProperty or DatatypeProperty definitions are stored in the ebXML registry, they can be retrieved through ebXML query facilities by the user. However, providing some stored procedures for this purpose facilitates the direct access. We therefore propose the following stored procedure to be available in the registry which retrieves all the object properties of a given classification node:

```
CREATE PROCEDURE findObjectProperties($className) AS
BEGIN
SELECT A.id
FROM Association A, Name_ N, ClassificationNode C
WHERE A.associationType LIKE 'objectProperty' AND
      C.id = N.parent AND
      N.value LIKE $className AND
      A.sourceObject = C.id
END;
```

A similar stored procedure can be given to retrieve datatype properties of a given class.

4.1.2. OWL class hierarchies. When it comes to mapping OWL class hierarchies to ebXML class hierarchies, OWL relies on RDF Schema for building class hierarchies through the use of "rdfs:subClassOf" property and allows multiple inheritance. In ebXML, Class Hierarchy is achieved by the "parent" association. However it is not possible to associate a ClassificationNode with two parents. In other words an ebXML Class hierarchy has a tree structure therefore is not readily available to express multiple inheritance, that is, there is a need for additional mechanisms to express multiple inheritance. We define a "subClassOf" property as an association for this purpose.

Consider the example:

```
<owl:Class rdf:ID="AirReservationServices">
 <rdfs:subClassOf rdf:resource="http://www.daml.org/services/
    owl-s/1.0/Profile.owl#Profile"/>
 <rdfs:subClassOf  rdf:resource="#AirServices"/>
</owl:Class>
```

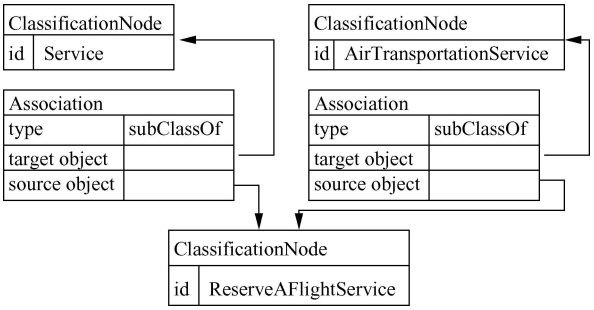


Figure 4. Representing an example "owl:subClassOf" property in ebXML registry.

Here, "AirReservationServices" service inherits both from "AirServices" service and OWL-S ServiceProfile class. Figure 4 shows how this is represented through ebXML RIM constructs. As presented in the figure, "AirReservationServices" ClassificationNode is associated with the "OWL-S Profile" and "AirServices" ClassificationNodes through the "target" and "source" object attributes of the newly created "subClassOf" ebXML Association.

Once we define such a semantics, we need the code to process the objects in the registry according to the semantics implied; that is, given a class, we should be able to retrieve all of its subclasses and/or all of its super classes. By making the required stored procedures available in the registry, this need can be readily served. For example, the following procedure finds all the immediate super classes of a given class:

```
CREATE PROCEDURE findSuperClasses($className) AS
BEGIN
SELECT C2.id
FROM Association A, Name_ N, ClassificationNode C1,
  ClassificationNode C2
WHERE A.associationType LIKE 'subClassOf' AND
      C1.id = N.parent AND
      N.value LIKE $className AND
      A.sourceObject = C1.id AND
      A.targetObject = C2.id
END;
```

Similar procedures can be provided to find all the superclasses of a given class (not only the immediate ones) as well as all its subclasses. The following procedure can then be used to retrieve all of the properties of a given class including the ones inherited from its super classes:

```
CREATE PROCEDURE findInheritedObjectProperties ($className) AS
SELECT A.id FROM Association A, ClassificationNode C WHERE
A.sourceObject=C.id AND
      A.associationType LIKE 'objectProperty' AND
      C.id IN (
            SELECT parent
```

```
            FROM Name_
            WHERE value LIKE $className
            UNION
            findSuperClasses($className)
            }
END;
```

4.1.3. OWL subPropertyOf. Since OWL properties are represented through ebXML associations, we define "rdfs:subPropertyOf" as an association between associations with a new association type of "subPropertyOf". The following procedure finds all the immediate super properties of a given property and similar procedures can be made available for all the super and subproperties:

```
CREATE PROCEDURE findSuperProperties($propertyName) AS
BEGIN
SELECT A3.id
FROM Association A1, Association A2, Association A3, Name_ N
WHERE A2.associationType LIKE 'subPropertyOf' AND
      A1.id = N.parent AND
      N.value LIKE $propertyName AND
      A2.sourceObject = A1.id AND
      A2.targetObject = A3.id
```

4.1.4. OWL equivalentClass, equivalentProperty and sameAs properties. In ebXML, the predefined "EquivalentTo" association (Table 2) expresses the fact that the source registry object is equivalent to target registry object. Therefore, "EquivalentTo" association is used to express "owl:equivalentClass", "equivalentProperty" and "sameAs" properties since classes, properties and instances are all ebXML registry objects.

Given a class, the following stored procedure retrieves all the equivalent classes:

```
CREATE PROCEDURE findEquivalentInstances($className) BEGIN SELECT
N.value FROM Service S, Name_ N WHERE S.id IN (
    SELECT classifiedObject
    FROM Classification
    WHERE classificationNode IN (
        SELECT id
        FROM ClassificationNode
        WHERE id IN (
            SELECT parent
            FROM Name_
            WHERE value LIKE $className
            )
        UNION
        SELECT A.targetObject
        FROM Association A, Name_ N, ClassificationNode C
        WHERE A.associationType LIKE 'EquivalentTo' AND
              C.id = N.parent AND
              N.value LIKE $className AND
              A.sourceObject = C.id
        )
```

```
    ) AND S.id=N.parent
END;
```

4.1.5. OWL transitive property. In OWL, if a property, P, is specified as transitive then for any x, y, and z: $P(x, y)$ and $P(y, z)$ implies $P(x, z)$. Transitive property can be defined as a new type of association in ebXML.

Consider the following example where we define the "succeeds" as a transitive property of "TravelWebService" class:

```
<owl:ObjectProperty rdf:ID="succeeds">
  <rdf:type rdf:resource="&owl;TransitiveProperty"/>
  <rdfs:domain rdf:resource="#TravelWebService"/>
  <rdfs:range rdf:resource="#TravelWebService"/>
</owl:ObjectProperty>
```

Assuming the following two definitions:

```
<TravelWebService rdf:ID="MyHotelAvailabilityService">
  <succeeds rdf:resource="#MyAirReservationService"/>
</TravelWebService>

<TravelWebService rdf:ID="MyInsuranceService">
  <succeeds rdf:resource="#MyHotelAvailabilityService"/>
</TravelWebService>
```

Since "succeeds" is a transitive property, it follows that "MyInsuranceService" succeeds "MyAirReservationService" although this fact is not explicitly stated.

To make any use of this transitive property in ebXML registries, coding is necesary to find out the related information. We provide the following stored procedure to handle this semantics: Given a class which is a source of a transitive property, this stored procedure retrieves not only the target of a given transitive property, but if the target objects have the same property, it also retrieves their target objects too.

```
CREATE PROCEDURE findTransitiveRelationships($className,
$propertyName) BEGIN SELECT A2.targetObject FROM Association A1,
Association A2, Name_ N1,Name_ N2, Name_ N3 WHERE
A1.associationType LIKE 'transitiveProperty' AND
      A1.id = N1.parent AND
      N1.value LIKE $propertyName AND
      A1.sourceObject = N3.parent AND
      N3.value LIKE $className AND
      A2.sourceObject = A1.targetObject AND
      A2.id = N2.parent AND
      N2.value LIKE $propertyName AND
      A2.associationType LIKE 'transitiveProperty'
UNION
SELECT A1.targetObject
FROM Association A1, Name_ N1, Name_ N3
WHERE A1.associationType LIKE 'transitiveProperty' AND
```

```
      A1.id = N1.parent AND
      N1.value LIKE $propertyName AND
      A1.sourceObject = N3.parent AND
      N3.value LIKE $className
END;
```

4.1.6. OWL inverseOf property. In OWL, if a property, $P1$, is tagged as the "owl:inverseOf" $P2$, then for all x and y: $P1(x, y)$ if $P2(y, x)$. Consider for example the "succeeds" property defined in Section 4.1.5. To denote that a certain Web service instance precedes another, we may define the "precedes" property as an inverse of the "succeeds" property as follows:

```
<owl:ObjectProperty rdf:ID="precedes">
  <owl:inverseOf rdf:resource="#succeeds"/>
</owl:ObjectProperty>
```

Then, by using the following stored procedure, we can find all the services that precede a given service by making use of its "succeeds" property.

```
CREATE PROCEDURE findInverseRanges($className, $propertyName)
BEGIN
SELECT C2.id
FROM Association A, Name_ N, Name_ N2, ClassificationNode C1,
      ClassificationNode C2
WHERE A.id=N2.parent AND
      N2.value LIKE $propertyName AND
      C1.id = N.parent AND
      N.value LIKE $className AND
      A.sourceObject = C1.id AND
      A.targetObject = C2.id
UNION
SELECT A3.sourceObject
FROM Association A1, Association A2, Association A3, Name_ N,
      NAME_ N2, ClassificationNode C1
WHERE A2.associationType LIKE 'inverseOf' AND
      A1.id = N.parent AND
      N.value LIKE $propertyName AND
      A2.sourceObject = A1.id AND
      A3.id=A2.targetObject AND
      C1.id = N2.parent AND
      N2.value LIKE $className AND
      A3.targetObject = C1.id
END;
```

4.1.7. OWL restriction. Another important construct of OWL is "owl:Restriction". In RDF, a property has a global scope, that is, no matter what class the property is applied to, the range of the property is the same. "owl:Restriction", on the other hand, has a local scope; restriction is applied on the property within the scope of the class where it is defined. The aim is to make ontologies more extendable and hence more reusable.

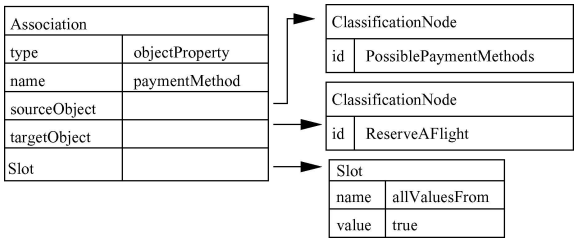


Figure 5. Representing an example OWL restriction in ebXML registry.

OWL provides the following language elements to indicate the type of restriction: *owl:allValuesFrom*, *owl:someValuesFrom*, *owl:hasValue*. An *owl:allValuesFrom* element defines the class of all objects for whom the values of property all belong to the class expression.

Consider the following example:

```
<owl:Class rdf:ID="AirReservationServices">
  <rdfs:subClassOf rdf:resource="&service"/>
  <rdfs:subClassOf rdf:resource= "#AirServices"/>
  <rdfs:subClassOf>
    <owlRestriction>
    <owl:onProperty rdf:resource="#paymentMethod"/>
    <owl:allValuesFrom rdf:resource= "#PossiblePaymentMethods"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Here "owl:Restriction" defines an anonymous class, that is the class of all things that satisfy this restriction. The restriction is that the property "paymentMethod" should get all of its values from the class "PossiblePaymentMethods". By defining "AirReservationServices" class as a subclass of this anonymous class, its "paymentMethod" property is restricted to the elements of the "PossiblePaymentMethods".

In ebXML class hierarchies, on the other hand, an association (which represents a property) is already defined in a local scope by associating two nodes of the class hierarchy. The type of the restriction can be expressed by special slot values. Figure 5 shows how the example above is represented through ebXML RIM constructs. Through the "allValuesFrom" slot added to the "paymentMethod" association, the type of the restriction is stated explicitly.

4.1.8. OWL class intersection. OWL provides the means to manipulate class extensions using basic set operators. In *OWL Lite*, only "owl:intersectionOf" is available which defines a class that consists of exactly all objects that belong to both of the classes. Consider the following example:

```
<owl:Class rdf:ID="AirReservationServices">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AirServices"/>
```

```
    <owl:Class rdf:about="#ReservationServices"/>
  </owl:intersectionOf>
</owl:Class>
```

In ebXML RIM "owl:intersectionOf" set operator can be expressed as follows:

- A new association type called "intersectionOf" is created.
- The classes constituting the intersection are represented as members of a Registry Package.
- The source object of the set operator is assigned as the sourceObject of the "intersectionOf" association.
- The target object of the "intersectionOf" association is set to be the newly created RegistryPackage.

The RIM representation of the OWL example presented above is presented in [12]. When such a representation is used to create a complex class in RIM, it becomes possible to infer that the objects classified by both of the classes constituting the intersection are also the instances of this complex class. The stored procedure retrieves the direct instances of the complex class and also the intersection of the instances of the member classes can be found in [12].

Table 4 provides a summary of how OWL language elements are mapped to ebXML class hierarchies. In this section, only some of these mappings are explained due to space limitations.

4.2. How to exploit OWL semantics for service discovery and composition in ebXML registry

In this section, in order to demonstrate the benefits of the proposed enhancements to the ebXML registry, we describe discovery of semantically enriched Web services through a "Web Service Discovery and Composition Definition Tool". This tool aids the user to find appropriate Web Services through automated service discovery in OWL aware ebXML registries and compose them into a workflow. Note that the emphasis of the work described is on semantic discovery of Web services not their semantic composition.

As presented in figure 6, the tool allows the ebXML classification hierarchies to be depicted graphically. When a user clicks on a node in the classification hierarchy, the generic properties of the service are retrieved from the registry and revealed to the user as depicted in figure 9. The user can fill in the desired properties of services she is looking for through this GUI. The tool queries the ebXML registry automatically to find the services that satisfy user constraints and the results are presented to the user in the second pane of figure 6. Then, a user can select from the presented service instances, can place the service to the appropriate slot in service choreography by dragging and dropping, and continues composition definition by adding control blocks when necessary. When the composition is finalized the BPEL4WS definition is created based on the WSDL files of the Web service instances retrieved from the ebXML registry.

Table 4. Mapping OWL ontologies to ebXML classification hierarchies without affecting the registry.

OWL	**ebXML**
owl:Class	ClassificationNode
individual	RegistryObject
rdf:Property	Association
rdfs:domain	sourceObject
rdfs:range	targetObject
owl:equivalentTo owl:samePropertyAs owl:sameAs	An association with a predefined association type of "EquivalentTo".
owl:differentFrom	An new association type of "differentFrom" is defined.
owl:AllDifferent owl:distinctMembers	A new association type "distinctMembers" is defined to add members to a Registry Package.
An association with a new association type is defined.	
rdfs:subClassOf owl:ObjectProperty owl:disjointWith owl:TransitiveProperty owl:FunctionalProperty owl:InverseFunctionalProperty owl:SymetricProperty	"subClassOf" "objectProperty" "disjointWith" "transitiveProperty" "functionalProperty" "inverseFunctionalProperty" "symetricProperty"
owl:DataTypeProperty	XML Schema datatypes are used by providing an external link from the registry.
rdfs:subPropertyOf owl:inverseOf	An association between associations with a new association type "subPropertyOf"/"inverseOf" is defined.
owl:intersectionOf	A registry package is created by associating the classes (i.e. the classification nodes) to be intersected through a new association type of "intersectionOf".
owl:Restriction	Since ebXML RIM associations have local scope, only the type of the Restriction needs to be specified.
A slot type is defined for the association representing a restriction.	
owl:allValuesFrom owl:someValuesFrom owl:hasValue	"allValuesFrom" "someValuesFrom" "hasValue"
An association with a new association type is defined.	
owl:cardinality owl:minCardinality owl:maxCardinality	"cardinality" "minCardinality" "maxCardinality"

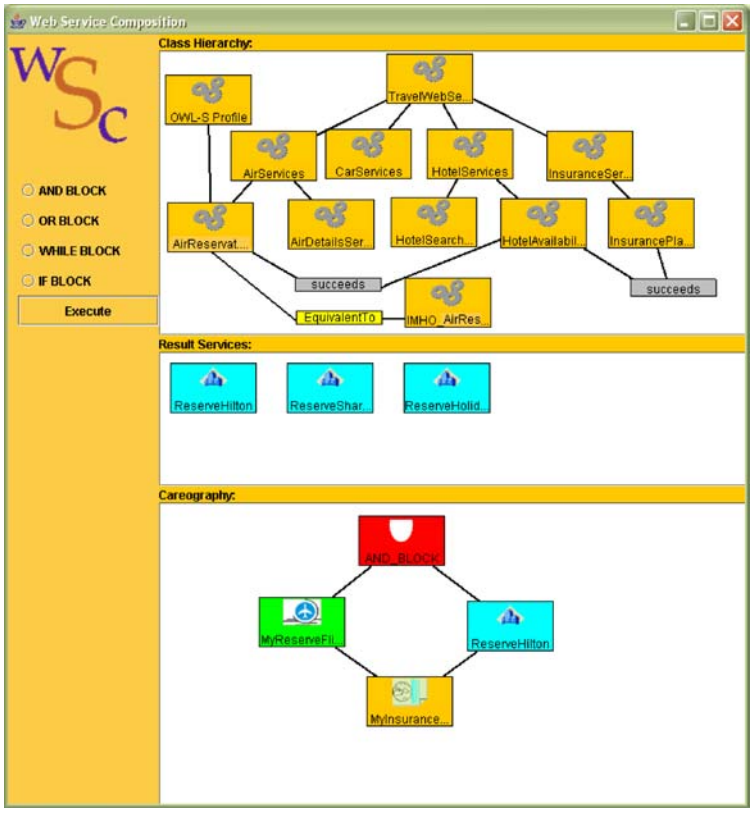


Figure 6. A snapshot of the GUI tool for semantics-based web service composition for ebXML registries.

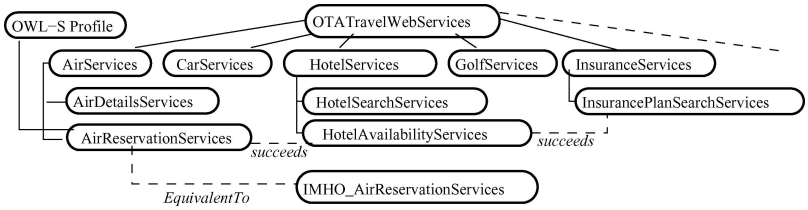


Figure 7. An example travel ontology.

In the following, we present an example to clarify how this semantic discovery mechanism works. Assume the travel ontology given in figure 7 is stored in the ebXML registry and is used in annotating travel Web services. Note that this ontology is based on "Open Travel Alliance" (OTA) specifications [30]. Assume further that a user wishes to organize a trip by first reserving a flight. By selecting the "AirReservationServices" ebXML ClassificationNode presented in the Web Service Composition tool, it becomes possible to query

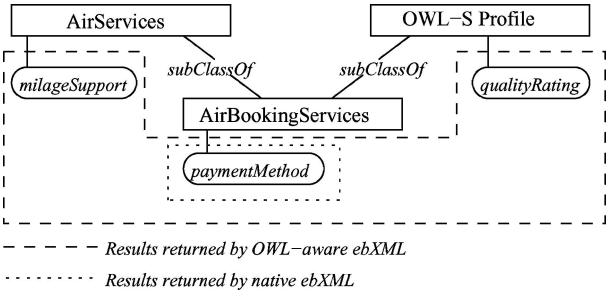


Figure 8. ebXML semantic support for class hierarchies.



Figure 9. A GUI tool to obtain the service property values from the user.

the services that are classified under the generic "AirReservationServices" node. In doing this, it is necessary to retrieve the properties of this class so that the user can provide her preferred values for the properties.

As presented in figure 7, "AirReservationServices" are defined as a subclass of both "OWL-S Profile" class and the "AirServices" class. In conventional ebXML, when a user submits a query to the ebXML registry to get the object properties of the "AirReservation-Services", only the immediate associations that are of type "objectProperty" are returned as presented in figure 8. However by exploiting the semantic capabilities of the OWL aware ebXML, the user can call the stored procedure "findInheritedObjectProperties" defined in Section 4.1.2 to retrieve the properties inherited from the parent classes too (figure 8).

These properties are shown to the user through the GUI depicted in figure 9. Once the user provides the preferred values, the instances satisfying these values are retrieved through the ebXML Filter query shown in figure 10 which is automatically issued through the tool. Note that while storing the Web service instances to ebXML registries, the values of their properties are represented through "slot" values.

```
<AdhocQueryRequest >
  <ResponseOption returnType = "LeafClass" returnComposedObjects = "true" />
   <FilterQuery>  <ServiceQuery>
    <ClassifiedByBranch>
        <ClassificationNodeQuery>  <NameBranch>  <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "Equal">AirReservationServices </StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </ClassificationNodeQuery>
      </ClassifiedByBranch>
    <SlotBranch>  <SlotFilter> <Clause>  <SimpleClause leftArgument = "name_">
        <StringClause stringPredicate = "Equal">paymentMethod</StringClause>
                </SimpleClause> </Clause> </SlotFilter>
        <SlotValueFilter> <Clause> <SimpleClause leftArgument = "value">
        <StringClause stringPredicate = "Contains">CreditCard</StringClause>
                </SimpleClause> </Clause> </SlotValueFilter>
    </SlotBranch>
    ...
    </ServiceQuery>   </FilterQuery> </AdhocQueryRequest>
```

Figure 10. An example filter query for retrieving the instances of the "AirReservationService".

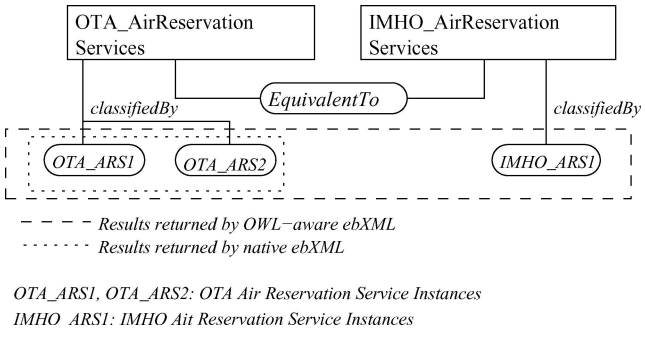


Figure 11. ebXML semantic support for equivalent classes.

Note further that the AirReservationServices node in the OTA Travel ontology (figure 7) is defined to be equivalent to "IMHO_AirReservationServices" (IMHO stands for "Interoperable Minimum Harmonise Ontology" which is a tourism ontology [22]). This relation is represented through the "EquivalentTo" type association of ebXML.

Without OWL semantic support, when the Filter Query presented in figure 10 is issued, the ebXML Query Manager will retrieve the services classified only by the AirReservationServices as presented in figure 11, using the SQL query presented in figure 12.

With OWL semantic support, our tool processes the semantics of the "EquivalentTo" property to retrieve the instances of the AirReservationServices by using the "findEquivalentInstances(AirReservationServices)" stored procedure defined in Section 4.1.4. Note that the use of these stored procedures is not restricted to our tool; any ebXML client can also use these stored procedures.

Assuming that the user chooses the "MyAirReservationService" instance among the Web services presented to her and discovering that 'AirReservationServices" has a "succeeds"

```
SELECT *  FROM Service WHERE id IN (
    SELECT classifiedObject FROM Classification
    WHERE classificationNode IN (
        SELECT id FROM ClassificationNode
        WHERE id IN (
            SELECT parent FROM name_
            WHERE value='AirReservationServices'
        )
    )
) AND id IN (
    SELECT parent FROM Slot
    WHERE name_='paymentMethod' AND
    value LIKE '%CreditCard%')
```

Figure 12. SQL query to retrieve the services classifed with "AirReservationServices".

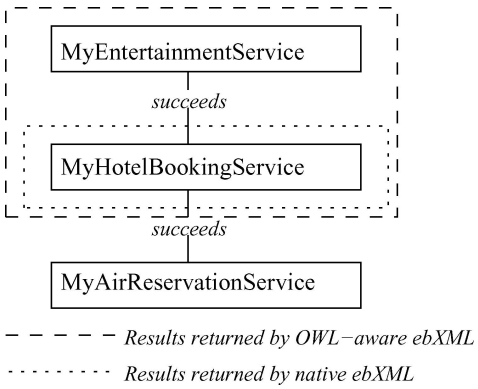


Figure 13. ebXML semantic support for transitive properties.

property, she may wish to consult to the ebXML for finding the "succeeding" services of this instance. Consider the example given in Section 4.1.5. When a user wishes to retrieve the "succeeding" services of the "MyAirReservationService" instance and issues a query to the ebXML registry, without OWL semantic support only "MyHotelAvailabilityService" instance will be returned as presented in figure 13, although "succeeds" has been declared to be transitive.

To be able to exploit the "transitivity" semantics, our tool uses the "findTransitiveRelationships(AirReservationServices,succeeds)" stored procedure defined in Section 4.1.5, which returns the "MyInsuranceService" instance additionally. Then the user can add these service instances to her choreography and to obtain the BPEL4WS definition.

4.3. Implementation status

A proof of concept implementation of the system is realized by using OASIS ebXML Registry Reference Implementation [18]. As an application server to host Web services to be accessed through SOAP, Apache Tomcat 4.1 [43] is used. The WSDL descriptions of the implemented services are generated through IBM Web Services Toolkit 3.2 (WSTK) [21], and BPEL4WS definitions are generated and executed using BPWS4J [20]. Finally OWL ontology is parsed with Jena 2.1 OWL parser [24].

5. Related work

In the early nineties, ontologies have been a research topic being addressed in a rather small research community. This has changed drastically in the late nineties by the insight that a conceptual, yet executable model of an application domain provides a significant value [17, 42]. The impact has increased with the Semantic Web initiative and the Web Ontology Language (OWL) [31].

The importance of semantics is also recognized in the Web services area and there have been several efforts to improve the semantics support for Web services [3, 4].

The need for extending the UDDI [44] registries with semantic capabilities has been addressed in the literature [8, 9, 34]. Note that UDDI registries use tModels to represent compliance with a taxonomy such as Universal Standard Products and Services Classification [46]. [8, 9] describe a mechanism to relate DAML-S ontologies with services advertised in the UDDI registries. [34] also addresses importing semantic to UDDI registries where DAML-S specific attributes such as *inputs, outputs* and *geographicRadius* are represented using tModel mechanisms of UDDI. An extended UDDI registry is also reported in [39] which allows to record user defined properties associated with a service and then to enable discovery of services based on these. In [40], the authors discuss adding semantics to WSDL using DAML+OIL ontologies. Their approach also uses UDDI to store these semantic annotations and search for Web services based on them.

As presented, although there has been a considerable amount of research for extending UDDI registries with OWL-S semantics for facilitating the discovery of Web Services, ebXML registries has not been studied much in this context. Since the semantic support provided by UDDI and ebXML registries differ considerably, it is not possible to repeat the previous work in UDDI for ebXML.

Related with ebXML, exploiting the native class hierarchies in ebXML registries for service discovery and composition is described in [10]. In [11], we present some initial ideas about enriching ebXML registries with OWL semantics. The work presented in this paper extends the ideas given in [11] and provides a complete model on how *OWL Lite* constructs can be stored to ebXML registries and queried to facilitate semantic discovery of Web Services.

An important effort in defining the semantics of Web Services is OWL-S [32] (previously DAML-S) which defined an upper ontology to describe service semantics. Related with exploiting DAML-S for service discovery and composition, some of the previous work use AI techniques to match the inputs and outputs of services requested and advertised. For example, [33] describes a matching engine to match advertised services with service requests, both defined in DAML-S. In [33], an advertisement matches a request when all the outputs of the request are matched by the outputs of the advertisement, and all the inputs of the advertisement are matched by the inputs of the request. In [47], DAML-S is extended to describe bioinformatics Web services and the services are matched by subsumption reasoning over the service descriptions.

In contrast to the approaches described above, we take a data management approach for service discovery part of the system by exploiting the metadata and the query mechanism of the ebXML registry. In our work, rather than having a user specifying the inputs and outputs of a request, the ebXML registry is queried through the enhanced registry constructs

to obtain the semantic information about the Web services. In this way service discovery reduces to querying the registry with the help of the ontology.

Medjahed et al. [26] proposes an ontology-based framework for the automatic composition of Web services. The authors present a technique to generate composite services from high-level declarative descriptions called "Composite Service Specification Language" (CSSL). For describing the semantics of Web Services, they extend WSDL with semantic capabilities. Composition plans are generated through a matchmaking algorithm according to composer's specifications in CSSL. Matchmaking algorithm takes a UDDI registry hosting Web Service WSDL definitions (extended with semantic constructs), retrieves a set of services from UDDI registry through the "service category" defined in CSSL, decides the semantic and syntactic "compasability" of Web services by comparing each Web service in this set with the preceding service in CSSL definition.

While this paper extends WSDL definitions with semantic constructs, in our approach domain ontologies are stored in ebXML registries, and semantics of Web services are defined through associating them to the ontology nodes in the registry. Furthermore, the work presented in [26] concentrates on the matchmaking of semantically enriched WSDL files and complements by our approach as follows: we show in detail how the semantic mechanisms of ebXML registries can be used to discover Web services semantically through registry queries. This phase provides the set of semantically suitable Web services prunning the unrelated services. After this step, a matchmaking algorithm as presented in [26] can be used to check their syntactic "composability".

Cardoso and Sheth [5] describes an algorithm to discover Web services and resolve heterogeneity among their interfaces and the workflow host. In this architecture, service discovery is achieved as follows: users can advertise the DAML-S definitions of their Web services to a registry, which is a service capability table where service descriptions are added, through a registry service. Through a discovery service, DAML-S profiles representing template of the queried service is sent to the system, then through a matchmaking algorithm presented in the paper, the services matched are presented to the user based on specific ranking criteria.

Our work complements this approach as follows: [5] proposes a generic semantic service discovery mechanism without addressing what a specific service registry may offer to help with service discovery. In our work, we show how to exploit the semantic constructs and query facilities of a specific service registry, namely, ebXML which is a widely adopted industry standard.

OASIS open source standards body [27] also has a number of standardization efforts aiming to support semantics within ebXML Framework. Some of these efforts will be presented in this section to put the work described in this paper into perspective. Note that the work described in this paper is also presented as a proposal to OASIS ebXML Semantic Content Management subcommittee which is working on possible semantic extensions to the registry.

There are several other key semantic requirements being addressed within the OASIS open source standards body [27]. This work is progressing through the committees including: The Business-Centric Methodology (BCM) Technical Committee (TC) [28], The ebXML Registry Semantic Content Management Sub-Committee (ebXMLR-SCM) [29]:

- *The business-centric methodology (BCM) technical committee (TC)* [28]: BCM addresses a proper interpretation of the business language semantics found in a SOA (Service Oriented Architecture) metadata framework/classification system which is essential for harnessing tacit knowledge and facilitating shared communications.
- *The ebXMLregistry semantic content management sub-committee (ebXMLR-SCM)* [29]: A key factor of the ebXMLR-SCM work towards semantic extensions of the Registry/Repository is the acknowledgment that the mapping of e-business artifacts to a semantic structure can employ many types of registry objects.

6. Conclusions and future work

This paper describes an engineering effort on how an ebXML registry can be made OWL aware. The work presented provides the foundation for OWL ontologies to be expressed in the registry. The representation of OWL semantics directly in the RIM enables the standard ebXML query facility to use stored procedures that can return a richer set of results based on explicit OWL constructs. As demonstrated, the queries of this type provide new capabilities to the ebXML client applications.

In this work, we use OWL Lite, since we are using ontologies to get knowledge through querying rather than reasoning. We investigate the possible ways of making the registry OWL aware and describe an approach that minimizes the changes on the ebXML specification.

There are two observations resulting from this experience:

- Ontologies can play two major roles: one is to provide a source of shared and precisely defined terms which can be used formalizing knowledge and relationship among objects in a domain of interest. The other is to reason about the ontologies. When an ontology language like OWL is mapped to a class hierarchy like the one in ebXML, the first role can directly be achieved. However, when we want to infer new information from the existing knowledge, we need reasoners. And reasoners can not directly run on the ebXML registry because all the registry information is stored in relational databases. Hence, there is a need to reconstruct the ontology from its representation in the ebXML registry.
- An ebXML registry client can use stored procedures that we have introduced to handle the OWL semantics. However, handling this semantics through the filter query in a transparent way to the user requires some modifications in the Query Manager Component of the registry. ebXML filter query, is designed to retrieve the registry objects as specified in the original RIM. It falls short to retrieve additional semantics introduced in this work.

  In "filter query", the user expresses what is to be retrieved from the registry as an XML message and the current syntax of ebXML query uses the conventional ebXML registry constructs. In order to retrieve extended semantics from in an OWL aware ebXML registry, through a "filter query", the Query Manager component needs to be extended.

  Consider the example defined in Section 4.1.8, where "AirReservationServices" is defined to be the intersection of the classes "AirServices" and "ReservationServices". When a user sends a Filter query to retrieve services classified by the "AirReservationServices" node, normally the ebXML Query Manager will return the services directly classified

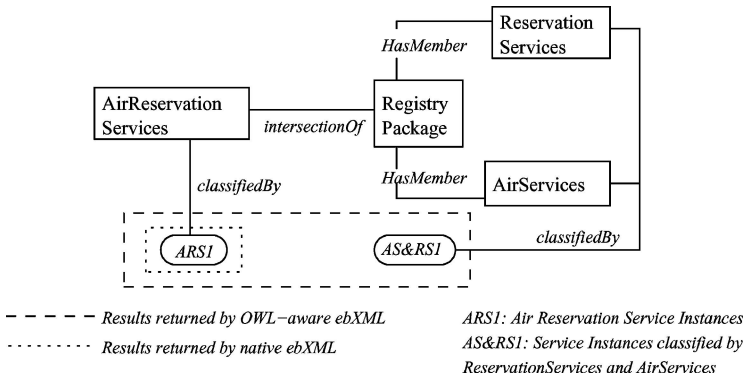


Figure 14. ebXML semantic support for class intersection.

by "AirReservationServices" node. However with OWL support it is possible to retrieve the services classified by both of the "AirServices" and "ReservationServices" at the same time, and thus retrieving the instances of "AirReservationServices", as presented in figure 14.

To handle such a semantics, the ebXML Query Manager should be updated to execute the "findInstances($className)" stored procedure defined in Section 4.1.8 whenever it receives such a filter query. In fact, the Query Manager needs to consider all such possibilities and this can only be handled through reasoning.

There are a number of public domain and commercial OWL reasoners such as [6, 13, 36]. As a future work, we intend to improve the Query Manager component with reasoning capabilities by exploiting one of the existing OWL reasoners.

References

1. G. Antoniou and F. Harmalen, Web ontology language: OWL, Handbook on Ontologies, Springer, 2004.
2. T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," Scientific American, May 2001.
3. C. Bussler, D. Fensel, T. Payne, and K. Sycara, "ISWC tutorial: semantic web services," http://www.daml.-ri.cmu.edu/tutorial/iswc-t3.html#2b
4. C. Bussler, D. Fensel, and A. Maedche, "A conceptual architecture for semantic web enabled web services," ACM Sigmod Record, vol. 31, no. 4, December 2002.
5. J. Cardoso and A. Sheth, "Semantic e-workflow composition," Journal of Intelligent Information Systems (JIIS), vol. 12, no. 3, 2003.
6. Cerebra OWL Reasoner, http://www.networkinference.com/Products/Cerebra_Server.html
7. DAML+OIL Reference Description, W3C Note, http://www.w3.org/2001/10/daml+oil, December 2001.
8. A. Dogac, I. Cingil, G.B. Laleci, and Y. Kabak, "Improving the functionality of UDDI registries through web service semantics," 3rd VLDB Workshop on Technologies for E-Services (TES-02), Hong Kong, China, August 23–24, 2002.
9. A. Dogac, G. Laleci, Y. Kabak, and I. Cingil, "Exploiting web service semantics: Taxonomies vs. ontologies," IEEE Data Engineering Bulletin, vol. 25, no. 4, December 2002.
10. A. Dogac, Y. Kabak, and G. Laleci, "A semantic-based web service composition facility for ebXML registries," in 9th International Conference of Concurrent Enterprising, Espoo, Finland, June 2003.

11. A. Dogac, Y. Kabak, and G. Laleci, "Enriching ebXML registries with OWL ontologies for efficient service discovery," in Proc. of RIDE'04, Boston, March 2004.
12. Example RIM representation of an Intersection, http://www.srdc.metu.edu.tr/∼banu/dapd/-RIMIntersectionExample Example Stored Procedure, http://www.srdc.metu.edu.tr/∼banu/dapd -/findInstancesStoredProcedure;
13. F-OWL Reasoner, http://fowl.sourceforge.net
14. ebXML, http://www.ebxml.org/
15. ebXML Registry Information Model v2.5, http://www.oasis-open.org/committees/regrep/documents/2.5/-specs/ebRIM.pdf
16. ebXML Registry Services Specification v2.5, http://www.oasis-open.org/committees/regrep/documents/2.5/-specs/ebRIM.pdf
17. D. Fensel, Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce, Springer, 2001.
18. freebXML Registry Open Source Project http://ebxmlrr.sourceforge.net
19. I. Horrocks, "DAML+OIL: A description logic for the semantic web," IEEE Data Engineering Bulletin, vol. 25, no. 1, March 2002.
20. IBM Business Process Execution Language for Web Services Java, http://www.alphaworks.ibm.com/tech/-bpws4j
21. IBM Web Services Toolkit (WSTK), http://alphaworks.ibm.com/tech/webservicestoolkit.
22. Minimum Harmonise Ontology, http://www.harmo-ten.info/index. -php?option= content&task= view&id-=5&Itemid=29
23. IST-2104-SATINE Project, http://www.srdc.metu.edu.tr/webpage/projects/satine/
24. Jena2 Semantic Web Toolkit, http://www.hpl.hp.com/semweb/jena2.htm
25. D. McGuinness and F. Harmelen, "OWL web ontology language overview," W3C Recommendation, February 2004, http://www.w3.org/TR/owl-features/
26. B. Medjahed, A. Bouguettaya, and A. Elmagarmid, "Composing web services on the semantic web," VLDB Journal, vol. 12, no. 4, 2003.
27. OASIS, http://www.oasis-open.org/home/index.php
28. OASIS Business-Centric Methodology (BCM) Technical Committee (TC), http://www.oasis-open.org/-committees/tc_home.php?wg_abbrev=bcm
29. OASIS ebXML Registry Semantic Content Management SC, http://www.oasis-open.org/apps/-org/workgroup/regrep-semantic/
30. Open Travel Alliance, http://www.opentravel.org/
31. OWL Web Ontology Language 1.0 Reference http://www.w3.org/TR/2002/WD-owl-ref-20020729/ref-daml
32. OWL-S, http://www.daml.org/services/daml-s/0.9/
33. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities," in Proc. of Intl. Semantic Web Conference, Sardinia, Italy, June 2002.
34. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Importing the semantic web in UDDI," in Web Services, E-Business and Semantic Web Workshop, 2002.
35. Registering web services in an ebXML Registry version 1.0. http://www.oasis-open.org/apps/org/workgroup/-regrep/download.php/1636/OASIS-Registry
36. Pellet OWL Reasoner, http://www.mindswap.org/2003/pellet/
37. RDF Schema: Resource Description Framework Schema Specification, W3C Proposed Recommendation, 1999, http://www.w3.org/TR/PR-rdf-schema.
38. RDF Syntax: Resource Description Framework Model and Syntax Specification, W3C Recommendation, 1999, http://www.w3.org/TR/REC-rdf-syntax.
39. A. ShaikhAli, O. Rana, R. Al-Ali, and D. Walker, "UDDIe: An extended registry for web services," Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference, Florida, January 2003.
40. K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller, "Adding semantics to web services standards," In proc. of ICWS, 2003.
41. M. Smith, C. Welty, and D. McGuinnes, "OWL web ontology language guide," W3C Recommendation, February 2004, http://www.w3.org/TR/owl-guide/

42. S. Staab and R. Studer, Handbook on Ontologies, Springer, 2004.
43. Tomcat, http://jakarta.apache.org/.
44. Universal Description, Discovery and Integration (UDDI), www.uddi.org
45. UN/CEFACT: United Nations Centre for Trade Facilitation and Electronic Business, http://www.unece.org/-cefact/index.htm.
46. Universal Standard Products and Services Classification (UNSPSC) http://eccma.org/unspsc
47. C. Wroe, R. Stevens, C. Goble, and A. Roberts, Greenwood, M., "A suite of DAML+OIL ontologies to describe bioinformatics web services and data," Intl. Journal of Cooperative Information Systems, to appear.