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
SIYAMED SINIR siyamed@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, USA

FARRUKH NAJMI farrukh.najmi@sun.com
Sun Micro Systems, USA

JEFF POLLOCK jeff.pollock@networkinference.com
Network Inference, USA

Abstract. In this paper, we address how ebXML registry semantics support can be further enhanced to make them OWL aware. There are basically three ways of achieving this: The first one is mapping OWL constructs to ebXML registry information model constructs without modifying the registry architecture and implementation. In this way, the semantic explicitly stored in the registry can be retrieved through querying; yet, the application program must contain additional code to process this semantics. The second approach is additionally providing predefined stored procedures in the registry for processing the OWL constructs. 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. Furthermore, with some minor modifications to the query management component of the registry, it is possible to handle this processing transparent to the user. The third approach is changing the ebXML registry to support OWL with full reasoning capabilities. However, this approach requires a dramatic change in the registry architecture and brings about the efficiency considerations of rule based systems.

Since our aim is to make the ebXML registry OWL aware by keeping the registry specification intact we take the second approach. 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.

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

* This work is supported in part by the European Commission, Project No: IST-1-002104-STP SATINE and by the Scientific and Technical Research Council of Turkey (TÜBİTAK), Project No: EEEAG 104E013

1. Introduction

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. 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) [28]. OWL is defined as three different sublanguages: *OWL Full*, *OWL DL* and *OWL Lite*, each geared towards fulfilling different requirements [22].

In this paper, we investigate how ebXML registries can be made OWL aware. In this way, we believe a mutually beneficial relationship is created between ebXML registries and the OWL ontology language: the former gains the ability to store OWL ontologies and the mechanisms provided by the ebXML registry proves very useful in associating the semantics defined in an OWL ontology with the registry objects. In this work, we use OWL Lite since we are using ontologies to get knowledge through querying rather than reasoning.

There are three alternatives to support OWL Lite ontologies through ebXML registries:

- 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.

- The code to process the OWL semantics can be stored in ebXML registry architecture through predefined procedures. For example, to find the the super classes of a given class (defined through a new association type of "subClassOf"), a 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 changing only the query manager component of the registry.

- The third approach is changing the ebXML registry architecture to support OWL with full reasoning capabilities. Reasoning entails the derivation of new data that is not directly stored in the registry. To deduce this data, rules need to be stored in the registry. However, this approach requires a dramatic 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.

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

Being OWL aware entails the following:

- Representing OWL constructs through ebXML constructs.
- Automatically generating ebXML constructs from the OWL descriptions and storing the resulting constructs into the ebXML registry.
- Querying the registry for enhanced semantics.

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 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 to make the registry OWL aware. Section 5 summarizes the related work. Finally, Section 6 concludes the paper.

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)

OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web [28]. OWL is derived from the DAML+OIL Web Ontology Language [8] and builds upon the Resource Description Framework [33, 34].

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, owl:hasValue*.

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 [20]. 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, owl:inverseFunctionalProperty*.

OWL provides three decreasingly expressive sublanguages [37]:

- *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* [22].
- *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.

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 registry and the repository via the ebXML Registry API as defined in [17]. The API has two main interfaces:

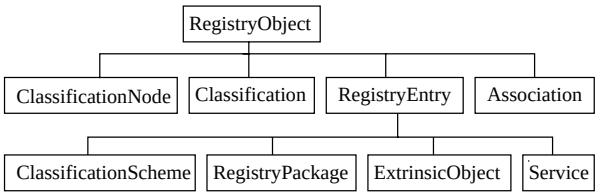

Figure 2. A Part of the ebXML RIM Class Hierarchy

- LifeCycleManager is the interface responsible for all object lifecycle management requests
- QueryManager 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. Client use 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
- 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 [16] 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 [16] related with storing metadata information. The main features of the information model includes:

- RegistryObject: 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 (a.k.a taxonomy) and ClassificationNodes which represent individual taxonomy element. A ClassificationScheme defines a tree structure made up of "ClassificationNode"s. 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 [16] as shown in Table 1. 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 server 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.

Name	**Description**
RelatedTo	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.

Table 1. Predefined Association Types in ebXML Registries

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

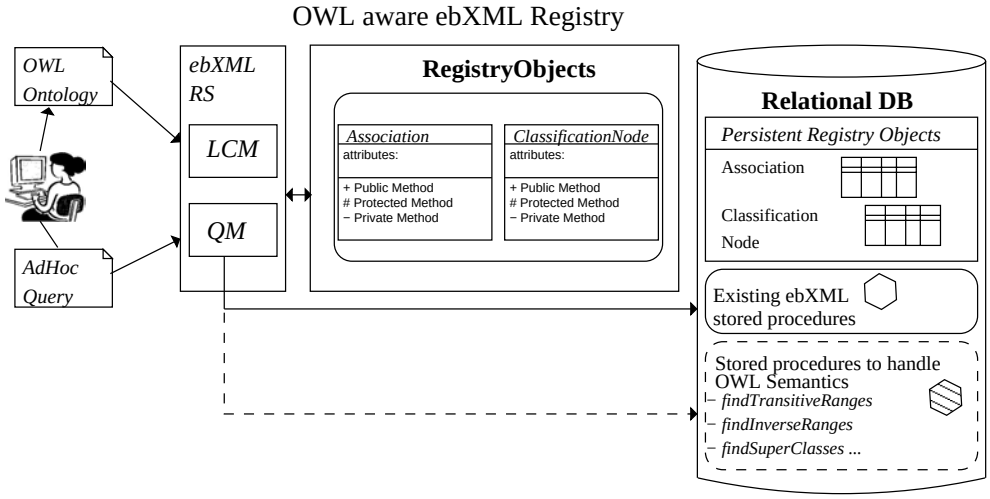

Figure 3. Enhancements to the ebXML Registry Architecture

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 is presented in Section 4.

Table 2. ebXML Relational Schemas

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

- 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 [21], 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 the stored procedures to process the OWL semantics introduced in Section 4. The stored procedures can be invoked directly by the user, or they can be invoked transparent to the user by slightly modifying the Query Manager. The modifications required in the query manager are as follows: the query manager while mapping the filter query to the SQL query should take the OWL semantics into account to use these stored procedures when necessary.

The enhanced architecture is shown in Figure 3. The OWL constructs are represented entirely through ebXML constructs by only extending the predefined associations which is allowed by the registry architecture. Hence there are no changes in the relational database schemas. We provide additional stored procedures to handle the OWL semantics. A user can handle the additional 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. If we wish 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 relared stored procedures we have introduced when necessary.

4. Providing OWL Lite Support to ebXML Registries

In this section, we first describe how OWL constructs can be represented in ebXML registry information model constructs. We then provide the stored procedures to make use of the additional semantics introduced. The stored procedures are defined using the ebXML relational schema specifications. The part of the schemas used in the examples are given in Table 2.

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 the ebXML registry and then give the required stored procedures to process this semantics.

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 (DatatypeProperty) in ebXML, we define a new type of association called "ObjectProperty" ("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 = 'promotion'associationType = 'ObjectProperty' sourceObject =
'City' targetObject = 'Airport' >
</rim:Association>
```

Consider the following example which defines an datatype property "hasPrice" whose domain is the "ReserveAFlight" 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="#ReserveAFlight"/>
  <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 = 'ReserveAFlight'
   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 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. An ebXML Class hierarchy has a tree structure, and 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:

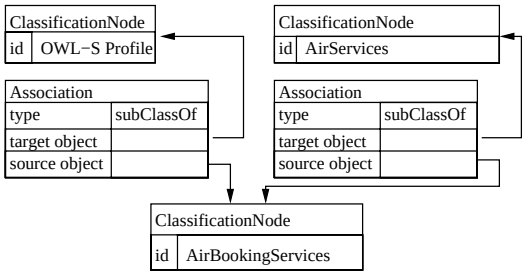

Figure 4. Representing an Example "owl:subClassOf" Property in ebXML Registry

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

Here, "AirBookingServices" service inherits both from "AirServices" service and OWL-S ServiceProfile class. Figure 4 shows how this is represented through ebXML RIM constructs.

Once we define such a semantics, we need the code to process the objects in the registry according to the 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 1) 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). Transivite 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="MyHotelBookingService">
  <succeeds rdf:resource="#MyAirReservationService" />
</TravelWebService>

<TravelWebService rdf:ID="MyEntertainmentService">
  <succeeds rdf:resource="#MyHotelBookingService" />
</TravelWebService>
```

Since "succeeds" is a transitive property, it follows that "MyEntertainmentService" 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 has 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) iff P2(y,x). Consider for example the "succeeds" property defined in

Section 4.1.5. To denote that a certain Web service instance preceeds another, we may define the "preceeds" property as an inverse of the "succeeds" property as follows:

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

Then, by using the following stored procedure, we can find all the services that preceeds a given service by making use of the "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:Resriction", 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. 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="ReserveAFlight">
  <rdfs:subClassOf rdf:resource="&service"/>
  <rdfs:subClassOf rdf:resource=
   "#AirTransportationService"/>
  <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 "ReserveAFlight" service 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.

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 do not 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 RegistryPackage.
- The source object of the set operator is assigned as the souceObject of the "intersectionOf" association.

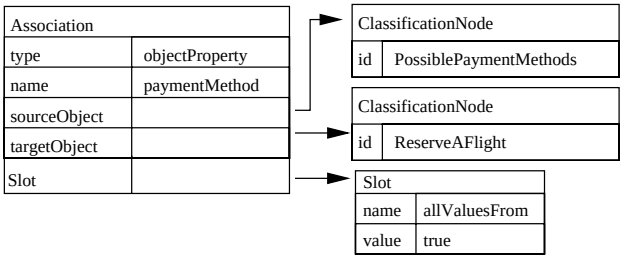

Figure 5. Representing an Example OWL Restriction in ebXML Registry

- 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 as follows:

```
<rim:ClassificationNode id = 'AirServices'  parent= 'TravelServices'>
    <rim:Name> <rim:LocalizedString value = 'AirServices' /> </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode id = 'ReservationServices'  parent= 'TravelServices'>
    <rim:Name> <rim:LocalizedString value = 'ReservationServices' /> </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode id = 'AirReservationServices'  parent= 'TravelServices'>
    <rim:Name> <rim:LocalizedString value = 'AirReservationServices' /> </rim:Name>
</rim:ClassificationNode>

<rim:RegistryPackage id = 'RP-AirServicesANDReservationServices'>
    <rim:Name> <rim:LocalizedString value = 'RP-AirServicesANDReservationServices'/>
    </rim:Name>
</rim:RegistryPackage>

<rim:Association id = 'firstMember' associationType = 'HasMember'
    sourceObject = 'RP-AirServicesANDReservationServices' targetObject = 'AirServices' >
    <rim:Name> <rim:LocalizedString value = 'firstMember' /> </rim:Name>
</rim:Association>

<rim:Association id = 'secondMember' associationType = 'HasMember'
    sourceObject = 'RP-AirServicesANDReservationServices' targetObject = 'ReservationServices' >
    <rim:Name> <rim:LocalizedString value = 'secondMember' /> </rim:Name>
</rim:Association>

<rim:Association id = 'intersectionOf' associationType = 'intersectionOf'
    sourceObject = 'AirReservationServices' targetObject = 'RP-AirServicesANDReservationServices' >
    <rim:Name> <rim:LocalizedString value = 'intersectionOf' /> </rim:Name>
</rim:Association>
```

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 following stored procedure retrieves the direct instances of the complex class and also the intersection of the member classes:

```
CREATE PROCEDURE findInstances($className) AS
BEGIN
SELECT N1.value
FROM Name_ N1, Service S, (
    SELECT A.targetObject AS id
    FROM RegistryPackage R, Association A
    WHERE R.id=A.sourceObject AND
          A.associationType = 'HasMember' AND
          R.id IN (
            SELECT A.targetObject
        FROM Association A, Name_ N, ClassificationNode C
        WHERE A.associationType LIKE 'intersectionOf' AND
              C.id = N.parent AND
              N.value LIKE $className AND
              A.sourceObject = C.id
        )
    ) AS T1, (
    SELECT A.targetObject AS id
    FROM RegistryPackage R, Association A
    WHERE R.id=A.sourceObject AND
          A.associationType = 'HasMember' AND
          R.id IN (
            SELECT A.targetObject
        FROM Association A, Name_ N, ClassificationNode C
        WHERE A.associationType LIKE 'intersectionOf' AND
              C.id = N.parent AND
              N.value LIKE $className AND
```

```
            A.sourceObject = C.id
        )
    ) AS T2
WHERE S.id IN (
    SELECT classifiedObject
    FROM Classification
    WHERE classificationNode=T1.id
    INTERSECT
    SELECT classifiedObject
    FROM Classification
    WHERE classificationNode=T2.id
    )  AND T1.id!=T2.id AND
    N1.parent=S.id
UNION
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
            )
        )
    ) AND S.id=N.parent
END;
```

Table 3 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 Lite Semantics in ebXML Registry

In this section, we present some examples to demonstrate the advantages of making the ebXML registry OWL aware.

4.2.1. OWL Class Hierarchies

Assume that "AirBookingServices" 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 "AirBookingServices", only the immediate associations that are of type "objectProperty" will be returned as presented in Figure 6. However by exploiting the semantic capabilities of the OWL aware ebXML, the user can issue the stored procedure "findInheritedObjectProperties" defined in Section 4.1.2 to retrieve the properties inherited from the parent classes too.

4.2.2. OWL Equivalent Classes

Assume there are more than one classification node in the ebXML registry for classifying hotel reservation services, namely "OTA_HotelReservationService" and

OWL	ebXML
owl:Class	ClassificationNode
rdf:Property	Association
rdfs:domain	sourceObject
rdfs:range	targetObject
owl:equivalentTo owl:samePropertyAs owl:sameAs	An association with a predefined association type of "EquivalentTo".
An association with a new association type is defined.	
rdfs:subClassOf	"subClassOf"
owl:ObjectProperty	"objectProperty"
owl:disjointWith	"disjointWith"
owl:TransitiveProperty	"transitiveProperty"
owl:FunctionalProperty	"functionalProperty"
owl:InverseFunctionalProperty	"inverseFunctionalProperty"
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:unionOf	A registry package is created by associating the classes (i.e. the classification nodes) whose union is to be taken, through a new association type of "unionOf".
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	"allValuesFrom"
owl:someValuesFrom	"someValuesFrom"
owl:hasValue	"hasValue"

Table 3. Mapping OWL Ontologies to ebXML Classification Hierarchies without Affecting the Registry

"IMHO_HotelReservationService"; and these are declared to be equivalent using the "EquivalentTo" type association of ebXML.

Without OWL semantic support, when the user issues the Filter Query presented in Figure 7, the ebXML Query Manager will retrieve the services classified by only

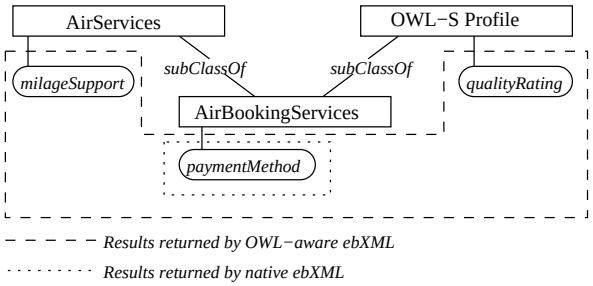

Figure 6. ebXML Semantic support for Class Hierarchies

```
<FilterQuery>
  <ServiceQuery>
    <ClassifiedByBranch>
      <ClassificationNodeQuery>
        <NameBranch>
         <LocalizedStringFilter>
           <Clause>
             <SimpleClause leftArgument = "value">
                <StringClause stringPredicate = "Equal">
                     OTA_HotelReservationService</StringClause>
             </SimpleClause>
           </Clause>
         </LocalizedStringFilter>
        </NameBranch>
      </ClassificationNodeQuery>
    </ClassifiedByBranch>
  </ServiceQuery>
</FilterQuery>
```

Figure 7. Filter Query to retrieve the services classifed with "OTA_HotelReservationService"

the OTA_HotelReserationService as presented in Figure 8, using the SQL query presented in Figure 9.

OWL semantic support can be added to ebXML Query Manager so that it can process the semantics of the "EquivalentTo" property to retrieve the instances of the IMHO_HotelReservationService transparent to the user. In this case Query manager can exploit the "findEquivalentInstances(OTA_HotelReservationService)" stored procedure defined in section 4.1.4. Alternatively the user himself can issue the "findEquivalentInstances(OTA_HotelReservationService)" without modifying the Query Manager, however in this case the user should be aware of the stored procedures designed for this purpose.

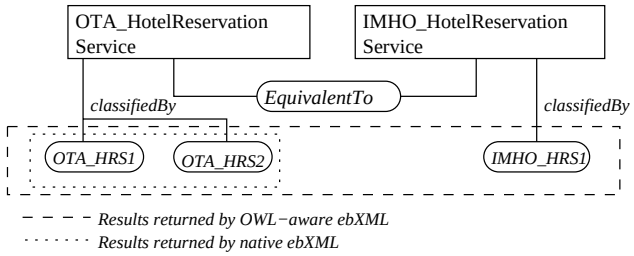

Figure 8. ebXML Semantic support for Equivalent Classes

```
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 = 'OTA_HotelReservationService' ) ) )
```

Figure 9. SQL query to retrieve the services classifed with "OTA_HotelReservationService"

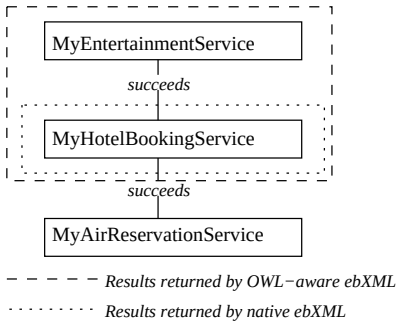

Figure 10. ebXML Semantic support for Transitive and inverseOf Properties

4.2.3. OWL Transitive and inverseOf Properties

Consider the example given in Section 4.1.5. When a user wishes to retrieve the "succeding" services of the "MyAirReservationService" instance and issues a query to the ebXML registry, without OWL semantic support only "MyHotelBookingSer-

vice" will be returned as presented in Figure 10, although "succeeds" has been declared to be transitive.

To be able to exploit the "transitivity" semantics, the user can use the "findTransitiveRelationships(MyAirReservationService,succeeds)" stored procedure, which will return the "MyEntertainmentService" instances additionally.

Similarly if the "precedes" property is declared to be the "inverseOf" the "succeeds" property, although a "precedes" relation has not been defined between the "MyHotelBookingService" and "MyAirReservationService"; if the user issues the "findInverseRanges(MyHotelBookingService, precedes)" stored procedure defined in Section 4.1.6, the ebXML Query Manager can return "MyAirReservationService" when the user asks the "preceding" services of the "MyHotelBookingService" service.

As mentioned earlier, to be able to exploit the semantics stored in the ebXML registry directly, the user should be aware of the "stored queries " proposed in this paper. The OWL semantics can be exploited transparently by the ebXML Query Manager as follows: whenever it receives a Filter Query from the user, it may execute the necessary "stored procedures". However in ebXML, filter queries are designed to retrieve the registry objects as specified in the original RIM. Therefore, Query Manager component of the ebXML registry may be modified to handle this additional semantics.

4.2.4. OWL Class Intersection

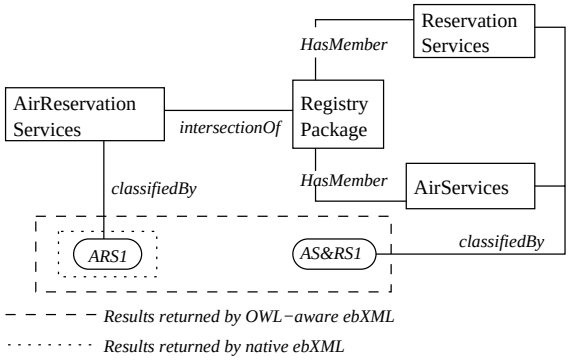

Figure 11. ebXML Semantic support for Class Intersection

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, (similar to the one presented in Figure 7), normally the ebXML Query Manager will return the services directly classified 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 11.

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.

4.3. How to Exploit OWL Lite Semantics for Service Discovery and Composition in ebXML Registry

TBD

5. Related Work

In the early nineties, ontologies have been a research topic being addressed in a rather small research community. This changed drastically in the late nineties by the insight that a conceptual, yet executable model of an application domain provides a significant value [38, 18]. The impact has increased with the Semantic Web initiative and the Web Ontology Language (OWL) [28].

The importance of semantics, being recognizing in the Web services area, there has been several efforts to improve the semantics support for Web services. OWL-S [29] defined an upper ontology to describe service semantics.

The need for extending the UDDI registries with semantic capabilities has been addressed in the literature [11, 12, 31]. UDDI registries use tModels to represent compliance with a taxonomy such as Universal Standard Products and Services Classification [39] and ebXML registries use classification hierarchies to associate semantics with registry items.

In [31], DAML-S specific attributes such as *inputs, outputs* and *geographicRadius* are represented using tModel mechanisms of UDDI. [11] and [12] also address importing semantic to UDDI registries where a mechanism is proposed to relate DAML-S ontologies with services advertised in the UDDI registries. 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 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, [30] describes a matching engine to match advertised services with service requests, both defined in DAML-S. In [30], 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. A request is matched against all the ad-

vertisements recorded in the registry and a degree of match is calculated. The matching engine described can draw an inference between inputs and outputs of the advertisement and requests, on the basis of the ontologies available in the registry [30]. The effiency of the matching engine is currently under testing. The work does not address the possible mechanisms a registry might have to help with the discovery of the services.

In [23] and [24], a logic-based agent technology is presented for service composition predicated on the use of reusable, task specific, high-level generic procedures and user-specific customizing constraints. The work is realized by using ConGolog and Prolog. ConGolog is an agent programming language built on top of situation calculus which in turn, is a logical language for reasoning about action and change. The generic procedures written in ConGolog capture what to do but not but not exactly how. As such it is deductively instantiated in the context of the agent's knowledge base, which includes properties of the agent and its user, properties of the specific services and the state of the world. The instantiation is performed by deductive machinery [24].

In [25], applicability of the DAML-S profile, process model, and grounding ontologies to the Web service lifecycle. In [41], DAML-S is extended to describe bioinformatics Web services and the services are matched by subsumption reasoning over the service descriptions.

[10] describes how to search for information and services on the Web when the Web pages are marked up through DAML. In this DAML-enabled search architecture, the author of a DAML-annotated Web page needs to provide sufficient information to link the services provided by the Web page with the concepts in an appropriate ontology.

Extending the UDDI registries with semantic capabilities is addressed in [11, 12], where we describe a mechanism to relate DAML-S ontologies with services advertised in the UDDI registries. [31] 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. The matching engine implemented is based on the algorithm described in [30]. An extended UDDI registry is reported in [35] which allows to record user defined properties associated with a service and then to enable discovery of services based on these. In [36], 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.

Exploting the class hierarchies in ebXML registries for service discovery and composition is described in [13, 14].

[3] proposes expanding the conceptual framework that underlies the semantic Web by connecting intelligent agents to Web services and hence giving them more behaviour.

In [5], a conceptual architecture called Web Service Modeling Framework (WSMF) is described based on the principles of strong decoupling and mediation of services.

A tutorial on semantic of Web services is available at [4] where a detailed overview of Web Service Modeling Framework (WSMF) is given. The authors state that Web Services on the Semantic Web are similar to Open Multi-Agent Systems and handle mediation through a set of middle agents such as Broadcaster Middle Agents, Yellow Pages Middle Agents, Matchmaker Middle Agent, and Recommender Middle Agent.

[6] describes an algorithm to discover Web services and resolve heterogeneity among their interfaces and the workflow host.

In eFlow [7] a service engine is described that supports the definition and enactment of adaptive and dynamic composite services. A composite service is modeled by a graph (the flow structure), which defines the order of execution among the nodes in the process. eFlow supports the dynamic creation of composite service definitions by including in its model the notion of "generic service" node.

[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. For this purpose, they extend WSDL with semantic capabilities.

An insightful description of Web services are given in [27] where the authors put the Web service technologies into perspective in Business-to-Business interaction domain.

6. Conclusions and Future Work

This paper describes an engineering effort on how an ebXML registry can be made OWL aware. 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 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. For reasoning, an ontology language is mapped to a known logical formalism and automated reasoners that already exist for that formalism are used. 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 it repesentation in the ebXML registry which might not be very efficient. We conclude that in order to natively support reasoning through OWL in ebXML registries, the registry architecture must rely on a knowledge base not a relational database. Given the considerations about

the performance of the rule based systems, it is questionable whether this effort is worth trying.

- The stored procedures that we have introduced can be directly used to handle the OWL semantics by the registry client as they are; or through SQL interfaces of the registry.

  ebXML filter query, on the other hand, is designed to retrieve the registry objects as specified in the original RIM. It falls short to retrieve additional semantics introduced in this work. For example, it is not possible to directly retrieve the associations of type "ObjectProperty" of a given classification node. It is still possible to retrieve all the associations of a given type which can be further processed to obtain the associations related with a given classification node.

  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.

References

1. Antoniou, G., Harmalen, F., "Web Ontology Language: OWL", in Handbook on Ontologies, Springer, 2004.
2. Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web", Scientific American, May 2001.
3. Bryson, J.J., Martin, D.L., McIlraith, S.A., Stein, L. A., "Toward Behavioral Intelligence in the Semantic Web", IEEE Computer, Vol. 35, No. 11, November 2002.
4. Bussler, C., Fensel, D., Payne, T., Sycara, K., "ISWC Tutorial: Semantic Web Services", http://www.daml.ri.cmu.edu/tutorial/iswc-t3.html#2b
5. Bussler, C., Fensel, D., Maedche, A., "A Conceptual Architecture for Semantic Web Enabled Web Services", ACM Sigmod Record, Vol. 31, No. 4, December 2002.
6. Cardoso, J., Sheth, A., "Semantic e-Workflow Composition", Journal of Intelligent Information Systems (JIIS), Vol. 12, No. 3, 2003.
7. Casati, F., Shan, M-C, "Dynamic and Adaptive Composition of E-Services", Information Systems, Vol.6, No.3, 2001.
8. DAML+OIL Reference Description, W3C Note, http://www.w3.org/2001/10/daml+oil, December 2001.
9. DAML Services Coalition (A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng), "DAML-S: Semantic Markup for Web Services", in Proceedings of the International Semantic Web Working Symposium (SWWS), July 2001.
10. Denker, G., Hobbs, J. R., Narayan, S., Waldinger, R., "Accessing Information and Services on DAML-Enabled Web, Semantic Web Workshop, Hong Kong, China, 2001.
11. Dogac, A., Cingil, I., Laleci, G. B., Kabak, Y., "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.
12. Dogac, A., Laleci, G., Kabak, Y., Cingil, I., "Exploiting Web Service Semantics: Taxonomies vs. Ontologies", IEEE Data Engineering Bulletin, Vol. 25, No. 4, December 2002.
13. Dogac, A., Kabak, Y., Laleci, G., "A Semantic-Based Web Service Composition Facility for ebXML Registries", 9th International Conference of Concurrent Enterprising, Espoo, Finland, June 2003.

14. Dogac, A., Kabak, Y., Laleci, G., "Exploiting Web Service Semantics Through ebXML Registries and Software Agents", submitted for publication.
15. ebXML, http://www.ebxml.org/
16. ebXML Registry Information Model v2.5, http://www.oasis-open.org/committees/regrep/-documents/2.5/specs/ebRIM.pdf
17. ebXML Registry Services Specification v2.5, http://www.oasis-open.org/committees/-regrep/documents/2.5/specs/ebRIM.pdf
18. Fensel, D., Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce, Springer, 2001.
19. freebXML Registry Open Source Project http://ebxmlrr.sourceforge.net
20. Horrocks, I., "DAML+OIL: a Description Logic for the Semantic Web", IEEE Data Engineering Bulletin, Vol. 25, No. 1, March 2002.
21. Jena2 Semantic Web Toolkit, http://www.hpl.hp.com/semweb/jena2.htm
22. McGuinness, D., Harmelen, F., "OWL Web Ontology Language Overview", W3C Recommendation, February 2004, http://www.w3.org/TR/owl-features/
23. McIlraith, S. A., Son, T. C., Zeng, H., "Semantic Web Services", IEEE Intelligent Systems, March/April 2001, pp. 46-53.
24. McIlraith, S. A., Son, T. C., Zeng, H., "Mobilizing the Semantic Web with DAML-Enaled Web Services", Semantic Web Workshop 2001, Hongkong, China.
25. McIlraith, S. A., Martin, D. L., "Bringing Semantics to Web Services", IEEE Intelligent Systems, Vol.18, No.1, 2003.
26. Medjahed, B., Bouguettaya, A., Elmagarmid, A., "Composing Web services on the Semantic Web", VLDB Journal, Vol.12, No.4, 2003.
27. Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A., Elmagarmid, A., "Business-to-business interactions: issues and enabling technologies", VLDB Journal, Vol.12, No.1, 2003.
28. OWL Web Ontology Language 1.0 Reference http://www.w3.org/TR/2002/WD-owl-ref-20020729/ref-daml
29. OWL-S, http://www.daml.org/services/daml-s/0.9/
30. Paolucci, M., Kawamura, T., Payne, T., Sycara, K., "Semantic Matching of Web Services Capabilities", in Proc. of Intl. Semantic Web Conference, Sardinia, Italy, June 2002.
31. Paolucci, M., Kawamura, T., Payne, T., Sycara, K., "Importing the Semantic Web in UDDI", in Web Services, E-Business and Semantic Web Workshop, 2002.
32. Registering web services in an ebXML Registry version 1.0. http://www.oasis-open.org/-apps/org/workgroup/regrep/download.php/1636/OASIS-Registry TC - Registering web services in an ebXML Registry.doc
33. RDF Schema: Resource Description Framework Schema Specification, W3C Proposed Recommendation, 1999, http://www.w3.org/TR/PR-rdf-schema.
34. RDF Syntax: Resource Description Framework Model and Syntax Specification, W3C Recommendation, 1999, http://www.w3.org/TR/REC-rdf-syntax.
35. ShaikhAli, A., Rana, O., Al-Ali, R., Walker, D., "UDDIe: An Extended Registry for Web Services", Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference, Florida, January 2003.
36. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J., "Adding Semantics to Web Services Standards", In proc. of ICWS, 2003.
37. Smith, M., Welty, C., McGuinnes, D., "OWL Web Ontology Language Guide", W3C Recommendation, February 2004, http://www.w3.org/TR/owl-guide/
38. Staab, S., Studer, R., Handbook on Ontologies, Springer, 2004.
39. Universal Standard Products and Services Classification (UNSPSC) http://eccma.org/unspsc
40. Web Content Management using ebXML Registry http://ebxmlrr.sourceforge.net/-presentations/xmlEurope2004/04-02-02.pdf http://ebxmlrr.sourceforge.net/presentations/-xmlEurope2004/xmlEurope2004-webcm-ebxmlrr.ppt http://ebxmlrr.sourceforge.net/-presentations/xmlEurope2004/xmlEurope2004-webcm-ebxmlrr.sxi
41. Wroe, C., Stevens, R., Goble, C., Roberts. A., Greenwood, M., "A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data", Intl. Journal of Cooperative Information Systems, to appear.