ebXML Registry Profile for Web Ontology Language (OWL)

Version 1.0 Draft 2

Draft OASIS Profile, January 27, 2006

Document identifier:

regrep-owl-profile-1.0-draft 2

Location:

http://www.oasis-open.org/committees/regrep-semantic/documents/profile/regrep-owl-profile-1.0-draft-1.pdf

Editors:

Name	Affiliation
Asuman Dogac	Middle East Technical University, Software R&D Center, Turkey
Yildiray Kabak	Middle East Technical University, Software R&D Center, Turkey
Gokce B. Laleci	Middle East Technical University, Software R&D Center, Turkey

Contributors:

Name	Affiliation
Farrukh Najmi	Sun Micro Systems, USA
Carl Mattocks	ITIL Application Knowledge Management, USA
Jeff Pollock	Network Inference, USA
....	

Abstract:

This document defines the ebXML Registry profile for enhancing ebXML Registry with OWL semantics to make it OWL aware.

Status:

This document is an OASIS ebXML Registry Semantic Content Management Committee Working Draft Profile and the work by the Editors is realized within the scope of the IST 2104 SATINE Project sponsored by the European Commission, DG Information Society and Media, eBusiness Unit.

Committee members should send comments on this specification to the regrep-semantic@lists.oasis-open.org list. Others should subscribe to and send comments to the regrep-comment@lists.oasis-open.org list. To subscribe, send an email message to regrep-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the OASIS ebXML Registry TC web page (http://www.oasis-open.org/committees/regrep/).

1 Table of Contents

Illustration Index

232

Index of Tables

1 Introduction

This chapter provides an introduction to the rest of this document.

The ebXML Registry holds the metadata for the RegistryObjects and the documents pointed at by the RegistryObjects reside in an ebXML repository. The basic semantic mechanisms of ebXML Registry are classification hierarchies (ClassificationScheme) consisting of ClassificationNodes and the Association Types among RegistryObjects. Furthermore, RegistryObjects can be assigned properties through a slot mechanism and RegistryObjects can be classified using instances of Classification, ClassificationScheme and ClassificationNodes. 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 to their data and applications [StaabStuder]. One of the driving forces for ontologies is the Semantic Web initiative [LeeHendler]. As a part of this initiative, W3C's Web Ontology Working Group defined Web Ontology Language [OWL].

Naturally, there is lot to be gained from using a standard ontology definition language, like OWL, to express semantics in ebXML registries.

This document normatively defines the ebXML Registry profile for Web Ontology Language (OWL) Lite. More specifically, this document normatively specifies how OWL Lite constructs SHOULD be represented by ebXML RIM constructs **without causing any changes in the core ebXML Registry specifications [ebRIM], [ebRS].** Furthermore, this document normatively specifies the code to process some of the OWL semantics through parameterized (generic) stored procedures that SHOULD be made available from the ebXML Registry.

These predefined stored queries provide the necessary means to exploit the enhanced semantics stored in the Registry. Hence, an application program does not have to develop additional code to process this semantics. In this way, it becomes possible to retrieve not only explicit but also the implied 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.

Finally it worths noting that ontologies can play two major roles: One is to provide a source of shared and precisely defined terms which can be used in 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. Furthermore some implicit information can be obtained by predefined parameterized queries. However, when we want full reasoning power, we need reasoners. Yet, OWL reasoners can not directly run on the ebXML registry because all the registry information is not stored in OWL syntax.

The document is organized as follows:

- Chapter 1 provides an introduction to the rest of this document.
- Chapter 2 provides an overview of the Web Ontology Language.
- Chapter 3 provides an overview of the ebXML Registry standard.
- Chapter 4 specifies the mapping between Web Ontology Language constructs and ebXML Registry Information Model. The stored procedures needed for the enhanced semantics is also given in this chapter.
- Chapter 5 provides normative and informative references that are used within or relevant to this document.

1.1 Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in IETF RFC 2119 [RFC211].

The term *“repository item*” is used to refer to content (e.g., an XML document or a DTD) that resides in a repository for storage and safekeeping. Each repository item is described by a RegistryObject instance. The RegistryObject catalogs the RepositoryItem with metadata.

1.2 Conventions

Throughout the document the following conventions are employed to define the data structures used. The following text formatting conventions are used to aide readability:

- UML Diagrams

UML diagrams are used as a way to concisely describe information models in a standard way. They are not intended to convey any specific Implementation or methodology requirements.

- Identifier Placeholders

Listings may contain values that reference ebXML Registry objects by their id attribute. These id values uniquely identify the objects within the ebXML Registry. For convenience and better readability, these key values are replaced by meaningful textual variables to represent such id values. For example, the following placeholder refers to the unique id defined for the canonical ClassificationNode that defines the Organization ObjectType defined in [ebRIM]:

```
<id="${CANONICAL_OBJECT_TYPE_ID _ORGANIZATION}" >
```

1.3 Recommendations

In the current ebXML Registry implementation, when a stored query is submitted to the ebXML Registry, it is stored in the “AdhocQuery” relational table without validation:

AdhocQuery (id, lid, objectType, status, versionName, comment_, queryLanguage, query);

When a user tries to invoke this stored query through a AdhocQuery, ebRS parses the stored query and converts this stored query to the syntax acceptable by the underlying database. Furthermore currently ebRS supports the SQL 92 [SQL 92] standard which does not include the “recursion” mechanisms. Also, there seems to be problems in parsing queries involving UNION. Since some of the queries involved in this Profile requires recursion and UNION mechanisms of SQL, it may help if ebRS is extended to support SQL 99 standard [SQL 99].

2 OWL Overview

This chapter provides an overview of the Web Ontology Language [OWL]. Web Ontology Language [OWL] is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is derived from the DAML+OIL Web Ontology Language [DAML+OIL] and builds upon the Resource Description Framework [RDF].

OWL provides three decreasingly expressive sublanguages [McGuinness, Harmelen]:

- **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.
- **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 document, only OWL Lite constructs are considered and in the rest of the document, “OWL” is used to mean “OWL Lite” unless otherwise stated.

OWL describes the structure of a domain in terms of classes and properties.

The list of OWL Lite language constructs is as follows [McGuinness, Harmelen]:

2.1 RDF Schema Features

- Class (Thing, Nothing)
- rdfs:subClassOf
- rdf:Property
- rdfs:subPropertyOf
- rdfs:domain
- rdfs:range
- Individual

2.2 (In)Equality

- equivalentClass
- equivalentProperty
- sameAs
- differentFrom
- AllDifferent
- distinctMembers

2.3 Property Characteristics

- ObjectProperty
- DatatypeProperty
- inverseOf
- TransitiveProperty
- SymmetricProperty
- FunctionalProperty

- InverseFunctionalProperty

2.4 Property Restrictions

- Restriction
- onProperty
- allValuesFrom
- someValuesFrom

2.5 Restricted Cardinality

- minCardinality (only 0 or 1)
- maxCardinality (only 0 or 1)
- cardinality (only 0 or 1)

2.6 Class Intersection

- intersectionOf

2.7 Versioning

- versionInfo
- priorVersion
- backwardCompatibleWith
- incompatibleWith
- DeprecatedClass
- DeprecatedProperty

2.8 Annotation Properties

- rdfs:label
- rdfs:comment
- rdfs:seeAlso
- rdfs:isDefinedBy
- AnnotationProperty
- OntologyProperty

2.9 Datatypes

- xsd datatypes

3 ebXML Registry Overview

This chapter provides an overview of ebXML Registry Information Model [ebRIM] and an overview of the specific domain and/or application.

The [ebRIM] is the target for the mapping patterns defined by this document.

The information presented is informative and is not intended to replace the normative information defined by ebXML Registry.

3.1 Overview of [ebRIM]

This section is provided in the « Deployment Profile Template for ebXML V3 specs » and can be removed in a specific profile.

Normally only specifics topics needs to be developed here (but the profile editor can prefer to leave it)

This section summarizes the ebXML Registry Information Model [ebRIM]. This model is the target of the mapping defined in this document. The reader SHOULD read [CMRR] for a more detailed overview of ebXML Registry as a whole.

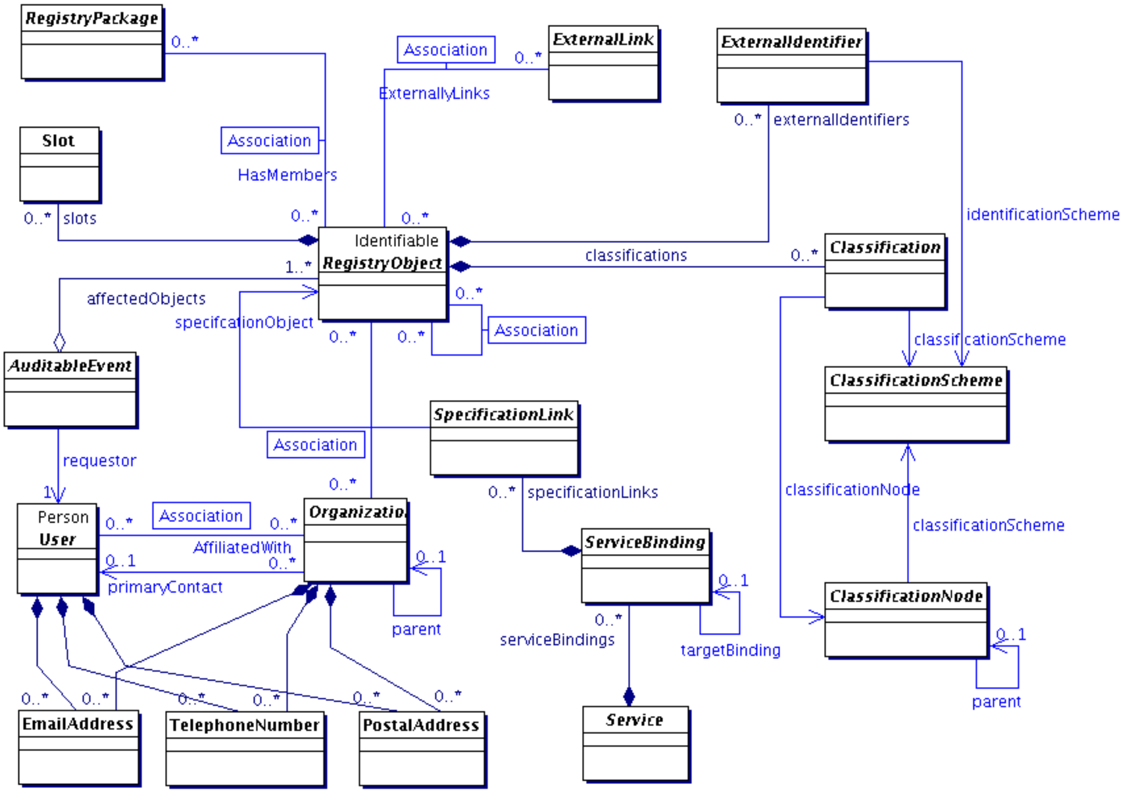

Figure 1: ebXML Registry Information Model, High Level Public View

The ebXML registry defines a Registry Information Model [ebRIM] that specifies the standard metadata that may be submitted to the registry. Figure 1 presents the UML class diagram representing the Registry Information Model. Figure 2, shows the inheritance relationships in among the classes of the ebXML Registry Information Model.

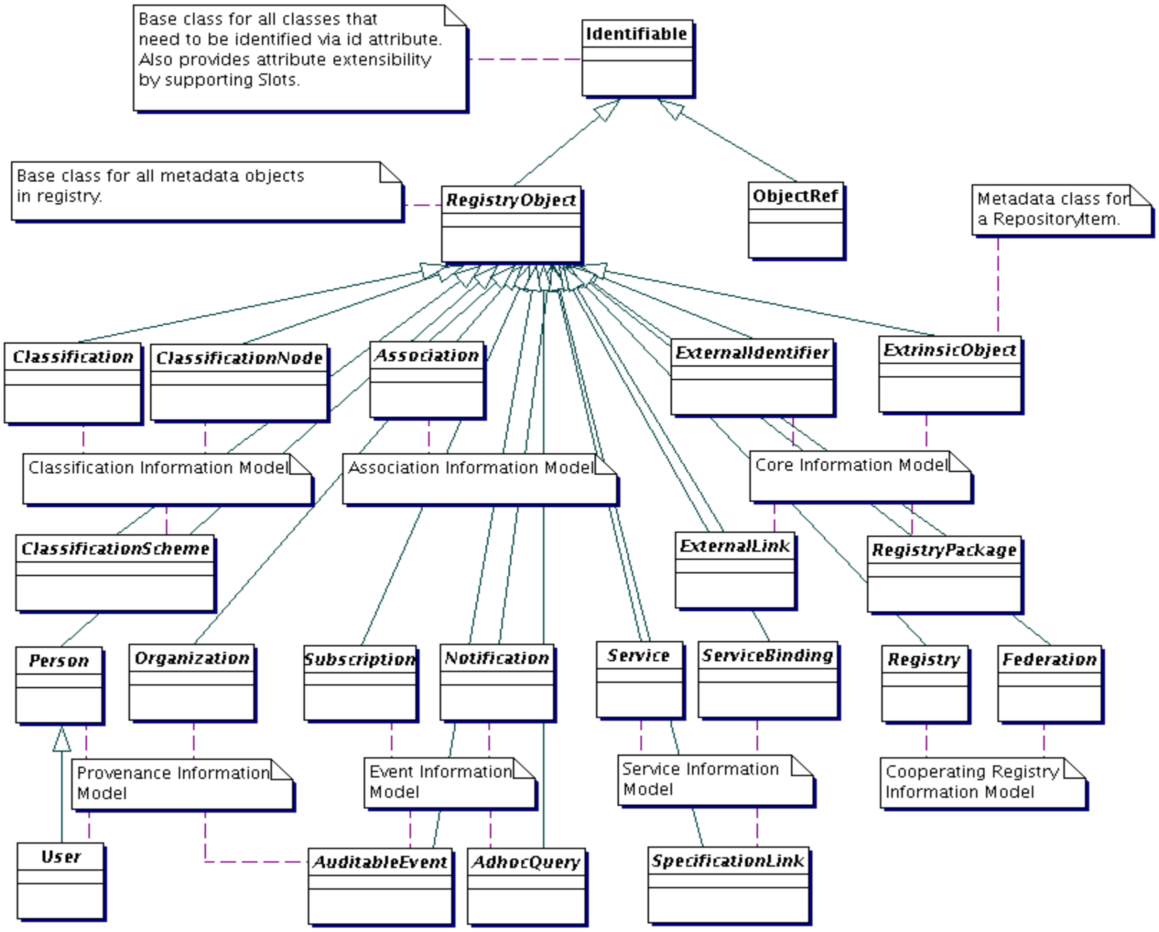
Base class for all classes that need to be identified via id attribute. Also provides attribute extensibility by supporting Slots.
Identifiable
Base class for all metadata objects in registry.
RegistryObject
ObjectRef
Metadata class for a RepositoryItem.
Classification
ClassificationNode
Association
ExternalIdentifier
ExtrinsicObject
Classification Information Model
Association Information Model
Core Information Model
ClassificationScheme
ExternalLink
RegistryPackage
Person
Organization
Subscription
Notification
Service
ServiceBinding
Registry
Federation
Provenance Information Model
Event Information Model
Service Information Model
Cooperating Registry Information Model
User
AuditableEvent
AdhocQuery
SpecificationLink

Figure 2: ebXML Registry Information Model, Inheritance View

The next few sections describe the main features of the information model.

3.1.1 RegistryObject

This is an abstract base class used by most classes in the model. It provides minimal metadata for registry objects. The following sections use the Organization sub-class of RegistryObject as an example to illustrate features of the model.

3.1.2 Object Identification

A RegistryObject has a globally unique id which is a UUID based URN:

```
<rim:Organization id="urn:uuid:dafa4da3-1d92-4757-8fd8-ff2b8ce7a1bf" >
```

Listing 1: Example of id attribute

The id attribute value MAY potentially be human friendly.

```
<rim:Organization id="uurn:oasis:Organization">
```

Listing 2: Example of human friendly id attribute

Since a RegistryObject MAY have several versions, a logical id (called lid) is also defined which is unique for different logical objects. However the lid attribute value MUST be the same for all versions of the same logical object. The lid attribute value is a URN that, as well for id attribute, MAY potentially be human friendly:

```
<rim:Organization id=${ACME_ORG_ID}
        lid="urn:acme:ACMEOrganization">
```

Listing 3: Example of lid Attribute

A RegistryObject MAY also have any number of ExternalIdentifiers which may be any string value within an identified ClassificationScheme.

```
<rim:Organization id=${ACME_ORG_ID}
        lid="urn:acme:ACMEOrganization">

        <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
               identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
               value="ACME"/>
      </rim:ExternalIdentifier>

</rim:Organization>
```

Listing 4: Example of ExternalIdentifier

3.1.3 Object Naming and Description

A RegistryObject MAY have a name and a description which consists of one or more strings in one or more local languages. Name and description need not be unique across RegistryObjects.

```
<rim:Organization id=${ACME_ORG_ID}
        lid="urn:acme:ACMEOrganization">

      <rim:Name>
        <rim:LocalizedString value="ACME Inc." xml:lang="en-US"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value="ACME is a provider of Java software."
               xml:lang="en-US"/>
      </rim:Description>

        <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
               identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
               value="ACME"/>
      </rim:ExternalIdentifier>
</rim:Organization>
```

Listing 5: Example of Name and Description

3.1.4 Object Attributes

For each class in the model, [ebRIM] defines specific attributes. Examples of several of these attributes such as id, lid, name and description have already been introduced.

3.1.4.1 Slot Attributes

In addition the model provides a way to add custom attributes to any RegistryObject instance using instances of the Slot class. The Slot instance has a Slot name which holds the attribute name and MUST be unique within the set of Slot names in that RegistryObject. The Slot instance also has a ValueList that is a collection of one or more string values.

The following example shows how a custom attribute named "*urn:acme:slot:NASDAQSymbol*" and value "*ACME*" MAY be added to a RegistryObject using a Slot instance.

```
<rim:Organization id=${ACME_ORG_ID}
        lid="urn:acme:ACMEOrganization">

```

```
      <rim:Slot name="urn:acme:slot:NASDAQSymbol">
        <rim:ValueList>
          <rim:Value>ACME</rim:Value>
        </rim:ValueList>
      </rim:Slot>

       <rim:Name>
        <rim:LocalizedString value="ACME Inc." xml:lang="en-US"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value="ACME makes Java. Provider of free Java 
software."
               xml:lang="en-US"/>
      </rim:Description>
       <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
               identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
               value="ACME"/>
      </rim:ExternalIdentifier>
</rim:Organization>
```

Listing 6: Example of a Dynamic Attribute Using Slot

3.1.5 Object Classification

Any RegistryObject may be classified using any number of Classification instance. A Classification instance references an instance of a ClassificationNode as defined by [ebRIM]. The ClassificationNode represents a value within the ClassificationScheme. The ClassificationScheme represents the classification taxonomy.

```
<rim:Organization id=${ACME_ORG_ID}
       lid="urn:acme:ACMEOrganization">
      <rim:Slot name="urn:acme:slot:NASDAQSymbol">
        <rim:ValueList>
          <rim:Value>ACME</rim:Value>
        </rim:ValueList>
      </rim:Slot>
       <rim:Name>
        <rim:LocalizedString value="ACME Inc." xml:lang="en-US"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value="ACME makes Java. Provider of free Java 
               software." xml:lang="en-US"/>
      </rim:Description>
       <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
               identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
               value="ACME"/>
      </rim:ExternalIdentifier>

       <!--Classify Organization as a Software Publisher using NAICS Taxonomy-->
      <rim:Classification id=${CLASSIFICATION_ID}
               classificationNode=${NAICS_SOFTWARE_PUBLISHER_NODE_ID}
               classifiedObject=${ACME_ORG_ID}>

</rim:Organization>
```

Listing 7: Example of Object Classification

3.1.6 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 [ebRIM] compliant. These canonical association types are defined as a *ClassificationScheme* called AssociationType. The SubmitObjectsRequest document of the AssociationType Classification scheme is available at:

http://www.oasis-open.org/committees/regrep/documents/3.0/canonical/SubmitObjectsRequest_AssociationTypeScheme.xml [ebRIM] allows this scheme to be extensible.

The following example shows an Association between the ACME Organization instance and a Service instance with the associationType of "OffersService". This indicates that ACME Organization offers the specified service (Service instance is not shown).

```
<rim:Association
        id=${ASSOCIATION_ID}
        associationType=${CANONICAL_ASSOCIATION_TYPE_OFFERS_SERVICE_ID}
        sourceObject=${ACME_ORG_ID}
        targetObject=${ACME_SERVICE1_ID}/>
```

Listing 8: Example of Object Association

3.1.7 Object References To Web Content

Any RegistryObject MAY reference web content that are maintained outside the registry using association to an ExternalLink instance that contains the URL to the external web content. The following example shows the ACME Organization with an Association to an ExternalLink instance which contains the URL to ACME's web site. The associationType of the Association MUST be of type "ExternallyLinks" as defined by [ebRIM].

```
<rim:ExternalLink externalURI="http://www.acme.com"
        id=${ACME_WEBSITE_EXTERNAL_ID}>
<rim:Association
        id=${EXTERNALLYLINKS_ASSOCIATION_ID}
        associationType=${CANONICAL_ASSOCIATION_TYPE_EXTERNALLY_LINKS_ID}
        sourceObject=${ACME_WEBSITE_EXTERNAL_ID}
        targetObject=${ACME_ORG_ID}/>
```

Listing 9: Example of Reference to Web Content Using ExternalLink

3.1.8 Object Packaging

RegistryObjects may be packaged or organized in a hierarchical structure using a familiar file and folder metaphor. RegistryPackage instances serve as folders while RegistryObject instances serve as files in this metaphor. A RegistryPackage instances groups logically related RegistryObject instances together as members of that RegistryPackage.

The following example creates a RegistryPackage for Services offered by ACME Organization organized in RegistryPackages according to the nature of the Service. Each Service is referenced using the ObjectRef type defined by [ebRIM].

```
<rim:RegistryPackage
        id=${ACME_SERVICES_PACKAGE_ID}>
        <rim:RegistryObjectList>
                <rim:ObjectRef id=${ACME_SERVICE1_ID}
                <rim:RegistryPackage
                        id=${ACME_PURCHASING_SERVICES_PACKAGE_ID}>
                        <rim:ObjectRef id=${ACME_ PURCHASING_SERVICE1_ID}
                        <rim:ObjectRef id=${ACME_ PURCHASING_SERVICE2_ID}
                </rim:RegistryPackage>
                <rim:RegistryPackage
                        id=${ACME_HR_SERVICES_PACKAGE_ID}>
                        <rim:ObjectRef id=${ACME_ HR_SERVICE1_ID}
                        <rim:ObjectRef id=${ACME_ HR_SERVICE2_ID}
                </rim:RegistryPackage>
        </rim:RegistryObjectList>
</rim:RegistryPackage>
```

Listing 10: Example of Object Packaging Using RegistryPackages

3.1.9 ExtrinsicObject

ExtrinsicObjects provide metadata that describes submitted content whose type is not intrinsically known to the registry and therefore MUST be described by means of additional attributes (e.g., mime type). Examples of content described by ExtrinsicObject include Collaboration Protocol Profiles, Business Process descriptions, and schemas.

3.1.10 Service Description

Service description MAY be defined within the registry using the Service, ServiceBinding and SpecificationLink classes defined by [ebRIM]. This MAY be used to publish service descriptions such as WSDL and ebXML CPP/A.

3.2 Overview of [ebRS]

The [ebRS] specification defines the interfaces supported by an ebXML Registry and their bindings to protocols such as SOAP and HTTP.

4 Representing OWL Constructs in ebRIM and Providing Processing Support for Additional Semantics

It is important to note that although the mapping described in this section is complex, this complexity is hidden from the ebXML registry user because the needed stored queries MUST already be available in the Registry as described in Chapter 6. As this profile aims to enhance ebXML registry semantics without causing any changes in the core ebXML Registry architecture specification [ebRIM], [ebRS], the stored queries proposed in this specification SHOULD be submitted to the ebXML Registry by using the Stored Query API of [ebRS].

The following ebRIM standard relational schema is used in coding the stored queries given in this section.

```
ClassScheme (id, home, lid, objectType, status, versionName, comment_,...);

ClassificationNode(accessControlPolicy, id, lid, home, objectType, code, parent, 
path,versionName, comment_...)

Association(accessControlPolicy, id, lid, home, objectType, associationType, 
sourceObject, targetObject, isConfirmedBySourceOwner,versionName, comment_ 
isConfirmedByTargetOwner,...)

Name_(charset, lang, value, parent,...)

Classification (id, objectType, lid, home, classificationNode, versionName, 
comment_, classificationScheme, classifiedObject, nodeRepresentation,...);

ExtrinsicObject (id, lid, home, objectType,...)
```

ebXML Registry Relations

Detailed explanation on how to represent some of the OWL Lite constructs in ebRIM is available from [Dogac, et. al.].

4.1 Representing RDF Schema Features in ebRIM

4.1.1 owl:Class → rim:ClassificationNode

An owl:Class MUST be mapped to a rim:ClassificationNode. For example, an OWL Class “City” which is a subclass of the Class “Country” can be mapped to ebRIM as follows: Two ClassificationNodes “City” and “Country” are defined where “City” is related to “Country” through the “parent” attribute of the ClassificationNode as shown in the following examples:

```
<owl:Class rdf:ID="City">
  <rdfs:subClassOf rdf:resource="#Country" />
</owl:Class>
```

Example owl:Class

```
<rim:ClassificationNode id='City' parent='Country' code='City'>
</rim:ClassificationNode>
```

Example Corresponding ebRIM construct ClassificationNode

4.1.2 rdf:Property → rim:Association Type Property

A new ebRIM Association Type called “Property” MUST be defined. The domain of an rdf:Property, rdfs:domain, is the sourceObject in this Association Type and the range of an rdf:Property which is rdfs:range, is the targetObject of the Association Type. Consider the following example which defines an rdf:Property instance called “hasAirport" whose domain is “City" and whose range is “Airport" classes:

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

Example rdf:Property

```
<rim:Association id='hasAirport' associationType='urn:oasis:names:tc:ebxml-
        regrep:AssociationType:Property'
  sourceObject= 'City'
  targetObject='Airport' >
</rim:Association>
```

Example: ebRIM construct Asssociation corresponding to rdf:Property

OWL specializes RDF Property to owl:ObjectProperty and owl:DatatypeProperty which are discussed in the sections 4.3.1 and 4.3.2.

4.1.3 rdfs:subPropertyOf → rim:Association Type subPropertyOf

In OWL, properties can be organized into property hierarchies by declaring a property to be a subPropertyOf another property. As shown in the following example, “creditCardPayment“ property may be a “subPropertyOf” the property “paymentMethods”:

```
<rdf:Property rdf:ID="creditCardPayment">
   <rdfs:subPropertyOf rdf:Resource="#paymentMethods"/>
</rdf:Property>
```

Example rdfs:subPropertyOf

A new ebXML RIM Association Type called “SubPropertyOf” MUST be defined to represent rdfs:subPropertyOf in ebRIM. Such a semantic enhancement brings the following processing need: given a property, it should be possible to retrieve all of its super properties as described in Section 6.1.

4.1.4 rdfs:subClassOf → rim:Association Type subClassOf

OWL relies on RDF Schema for building class hierarchies through the use of "rdfs:subClassOf" property and allows multiple inheritance. In ebXML, a class hierarchy is represented by a ClassificationScheme. A ClassificationScheme is constructed by connecting a ClassificationNodes to its super class by using the “parent" attribute of the ClassificationNode. However it is not possible to associate a ClassificationNode with more than one different super classes by using “parent” attribute. In other words, an ebXML Class hierarchy has a tree structure and therefore is not readily available to express multiple inheritance. There is a need for additional mechanisms to express multiple inheritance in ebXML RIM. Therefore, a new Association Type called "subClassOf" MUST be defined in the Registry.

In the following OWL example, "AirReservationServices" service inherits both from "AirServices" service and OWL-S ServiceProfile class.

```
<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>
```

Example rdfs:subClassOf

To express this semantics through ebXML RIM constructs, "AirReservationServices" ClassificationNode is associated both with the "OWL-S Profile" and "AirServices" ClassificationNodes through the "targetObject" and "sourceObject" attributes of the two instances of the newly created "subClassOf" ebXML Association

Type as shown in the following:

```
<rim:Association id='subClassOf1' associationType='urn:oasis:names:tc:ebxml-
       regrep:AssociationType:SubClassOf'
  sourceObject= 'AirReservationServices' targetObject='OWL-S Profile' >
</rim:Association>
<rim:Association id='subClassOf2' associationType='urn:oasis:names:tc:ebxml-
       regrep:AssociationType:SubClassOf'
  sourceObject= 'AirReservationServices' targetObject='AirServices' >
</rim:Association>
```

Once such a semantics is defined, there is a need to process the objects in the registry according to the semantics implied; that is, given a class, it should be possible to retrieve all of its subclasses and/or all of its super classes. By making the required adhoc queries available in the registry, this need can be readily served as described in Secion 6.2, 6.3, 6.4 and 6.5.

4.1.5 owl:Individual → rim:ExtrinsicObject

A class in OWL defines a group of individuals that belong together because they share some properties [McGuinness, Harmelen]. For example, "TravelService" class may have the property "paymentMethod" whose range may be "PossiblePaymentMethods" class as shown in the following example:

```
<owl:Class rdf:ID="TravelWebService">
</owl:Class>

<owl:ObjectProperty rdf:ID="paymentMethod">
  <rdfs:domain rdf:resource="#TravelWebService"/>
  <rdfs:range rdf:resource="#PossiblePaymentMethods"/>
</owl:ObjectProperty >
```

Example owl:Class example

In OWL, individuals are instances of classes. For example, an instance of "TravelWebService" class may be "MyTravelWebService". Properties may be used to relate one individual to another. For example, "MyTravelService" inherits "paymentMethod" property and this property may map to an instance of "PossiblePaymentMethods" class, such as "Cash" as shown in the following example:

```
<TravelWebService rdf:ID="MyTravelWebService">
<paymentMethod> Cash </paymentMethod>
</TravelWebService>
```

Example owl:Individual example

In ebXML Registry the class instances can be stored in the Registry or in the Repository. However, since ebXML philosophy is to store metadata in the Registry and the data (i.e., the instances) in the Repository, it may be more appropriate to store class instances in the Repository and describe their metadata through ExtrinsicObjects in the Registry.

4.2 Representing OWL (In)Equality Constructs in ebXML RIM

4.2.1 owl:equivalentClass, owl:equivalentPropery → rim:Association Type EquivalentTo

In ebXML, the predefined "EquivalentTo" Association Type expresses the fact that the source RegistryObject is equivalent to target RegistryObject. Therefore, "EquivalentTo" association MUST be used to express "owl:equivalentClass" and "owl:equivalentProperty" properties since classes and properties are all ebXML RegistryObjects.

The adhoc query for retrieving all the equivalent classes of a given ClassificatioNode is represented in Section 6.6. Additionally the adhoc query to retrieve all the equivalent properties (Association Type) of a given property (Association Type) is presented in Section 6.7

4.2.2 owl:sameAs → rim:Association Type sameAs

ebXML Registry contains the metadata of the objects stored in the repository. In other words, the instances are stored in repository and represented through “ExtrinsicObjects” in the registry.

owl:sameAs construct is used to indicate that two instances in a knowledge base are the same. This construct may be used to create a number of different names that refer to the same individual.

```
<rdf:Description rdf:about="#MyAirReservationService">
   <owl:sameAs rdf:resource="#THYAirReservationService"/>
</rdf:Description>
```

Example owl:sameAs

This translates into two "ExtrinsicObjects" in the ebXML registry to be the same. For this purpose a new Association Type called "sameAs" MUST be defined in the ebXML registry.

Furthermore, the adhoc query presented in Section 6.8 MUST be available in the registry to retrieve all the "ExtrinsicObjects" defined to be the same with a given ExtrinsicObject.

4.2.3 owl:differentFrom → rim:Association Type differentFrom

owl:differentFrom construct is used to indicate that two instances in a knowledge base are different from one another. Explicitly stating that individuals are different can be important in when using languages such as OWL (and RDF) that do not assume that individuals have one and only one name [McGuinness, Harmelen].

```
<rdf:Description rdf:about="#MyAirReservationService">
   <owl:differentFrom rdf:resource="#THYAirReservationService"/>
</rdf:Description>
```

Example owl:differentFrom

This translates into declaring two "ExtrinsicObjects" in the ebXML registry to be different from each other. For this purpose a new Association Type "differentFrom" MUST be defined in the ebXML registry to explicitly indicate that the sourceRegistryObject is different from the targetRegistryObject. The adhoc query presented in Section 6.9 can be used to process this semantics.

4.2.4 owl:AllDifferent

owl:AllDifferent is a special built-in OWL class, for which the property owl:distinctMembers is defined, which links an instance of owl:AllDifferent to a list of individuals. The AllDifferent construct is particularly useful when there are sets of distinct objects and when modelers are interested in enforcing the unique names assumption within those sets of objects [McGuinness, Harmelen].

The following example states that the three instances of the “WebService” collection are all different from one another:

```
<owl:AllDifferent>
   <owl:distinctMembers rdf:parseType="Collection">
       <WebService rdf:about="#MyCarService"/>
       <WebService rdf:about="#MyFlightService"/>
       <WebService rdf:about="#MyHotelService"/>
   </owl:distinctMembers>
</owl:AllDifferent>
```

Example owl:AllDifferentFrom

owl:AllDifferent SHOULD be represented in ebRIM as follows: the RegistryObjects under consideration SHOULD be grouped as a RegistryPackage called “Collection”. Then the RegistryObjects in the collection MUST be associated with this RegistryPackage with “hasMember” Association Type. One slot of the registry package MUST be used to indicate that all members are different.

IMPORTANT NOTE: When trying to submit the following “SubmitObjectsRequest”, we get the following unexpected error from the freebXML which implies that in the new Registry implementation it is not possible to associate “slots” with RegistryPackages which seems there is a bug in the software.

javax.xml.bind.UnmarshalException: Unexpected element {urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0}:Slot

The adhoc query presented in Section 6.10 can be used to process this semantics.

4.3 Representing OWL Property Characteristics in ebRIM

4.3.1 owl:ObjectProperty → rim:Association Type objectProperty

To represent OWL ObjectProperty in ebXML, a new type of Association called “ObjectProperty" MUST be defined. 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>
```

Example owl:ObjectProperty

```
<rim:Association id='hasAirport' associationType='urn:oasis:names:tc:ebxml-
regrep:AssociationType:ObjectProperty'
  sourceObject= 'City' targetObject='Airport' >
</rim:Association>
```

Example Corresponding ebRIM construct Asssociation

Once such objectProperty definitions are stored in the ebXML registry, they can be retrieved through ebXML query facilities by the user. The adhoc queries presented in Section 6.11 and 6.12 MUST be available in the registry to facilitate this access.

4.3.2 owl:DatatypeProperty → rim:Association Type DatatypeProperty

Similarly, to represent OWL DatatypeProperty in ebXML, a new Association Type called “DatatypeProperty" MUST be defined. Consider the following example which defines an datatype property “hasPrice" whose domain is the “AirReservationServices" and whose range is “XMLSchema nonNegativeInteger". How OWL XML Schema types are handled in ebXML RIM is described in Section 4.9.

```
<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>
```

Example owl:DatatypeProperty

The adhoc query presented in Section 6.14 MUST be available in the registry to facilitate the direct access to datatype properties of a given classification node.

4.3.3 owl:TransitiveProperty → rim:Association Type transitiveProperty

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) [McGuinness, Harmelen]. Transitive property is a subproperty of ObjectProperty and MUST be defined as a new Association Type called “transitiveProperty” in ebRIM.

Consider the following example where "succeeds" is defined 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>
```

Example owl:TransitiveProperty

Assume the following two definitions which declare three Web service instances from TravelWebService class where "MyHotelAvailabilityService" service succeeds "MyAirReservationService" and "MyInsuranceService" succeeds MyHotelAvailabilityService". Since "succeeds" is a transitive property, it follows that "MyInsuranceService" succeeds "MyAirReservationService" although this fact is not explicitly stated.

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

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

Example owl:TransitiveProperty instances

To make any use of this transitive property in ebXML registries, coding is necessary to find out the implied information. The adhoc query presented in Section 6.16 MUST be available in the registry to handle this semantics.

4.3.4 owl:inverseOf → rim:Association Type inverseOf

In OWL, one property may be stated to be the inverse of another property. If the property P1 is stated to be the inverse of the property P2, then if X is related to Y by the P2 property, then Y is related to X by the P1 property [McGuinness, Harmelen].

Consider, for example, the "succeeds" property defined in Section 4.3.3. To denote that a certain Web service instance precedes another during execution, 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>
```

Example owl:inverseOf Property

Assume that we want to find all the Web services which can succeed a given Web service. In such a case, we need not only find all the Web services which succeeds this given Web service, that is the target objects of "succeeds" Association instance, but we also need to find all the sourceObjects of the "precedes" Association instance since "precedes" is declared to be the "inverseOf" succeeds Association instance. This can be achieved through the adhoc query presented in Section 6.19.

4.3.5 owl:SymmetricProperty→ rim:Association Type SymmetricProperty

In OWL, if a property is symmetric, then if the pair (x,y) is an instance of the symmetric property P, then the pair (y,x) is also an instance of P [McGuinness, Harmelen]. Symmetric property is a subproperty of ObjectProperty in OWL. Consider the OWL class “WebService” and the “complements” symmetric property:

```
<owl:Class rdf:ID="WebService">
  <rdfs:subClassOf
       rdf:resource="http://www.w3.org/2000/01/rdfschema#Resource"/>
</owl:Class>
<owl:SymmetricProperty rdf:ID="complements">
       <rdfs:domain rdf:resource="#WebService"/>
       <rdfs:range rdf:resource="#WebService"/>
```

```
</owl:SymmetricProperty>
```

Example owl:SymmetricProperty

Given that HotelReservationWebService complements AirReservationWebService, it is possible to deduce that AirReservationWebService complements HotelReservationWebService.

owl:SymmetricProperty MUST be defined as a new type of Association in ebRIM called “SymmetricProperty”. Furthermore the adhoc query presented in Section 6.20 MUST be available in the Registry to retrieve symmetric Associations of a ClassificationNode.

4.3.6 owl:FunctionalProperty→ rim:Association Type FunctionalProperty

In OWL, if a property is a FunctionalProperty, then it has no more than one value for each individual (it may have no values for an individual) [McGuinness, Harmelen]. The range of a FunctionalProperty can be either an Object or a datatype. Consider, for example, the “hasPrice” Functional property which has a unique price:

```
<owl:DatatypeProperty rdf:ID="hasPrice">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#AirReservationServices"/>
  <rdfs:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema/nonNegativeInteger"/>
</owl:DatatypeProperty>
```

Example owl:FunctionalProperty

ebXML RIM MUST contain a new Association Type called “FunctionalProperty” to express this semantics. Furthermore the he adhoc query presented in Section 6.21 MUST be available in the Registry to retrieve functional Associations of a ClassificationNode.

4.3.7 owl:InverseFunctionalProperty→ rim:Association Type InverseFunctionalProperty

In OWL, if a property is inverse functional then the inverse of the property is functional. Thus the inverse of the property has at most one value for each individual [McGuinness, Harmelen].

As an example, the ObjectProperty “departsFrom” indicates that each flight originates from only one airport.

```
<owl:ObjectProperty rdf:ID="departsFrom">
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
  <rdfs:domain rdf:resource="#Airport"/>
  <rdfs:range rdf:resource="#Airport"/>
</owl:ObjectProperty>
```

Example owl:InverseFunctionalProperty

ebRIM MUST contain a new Association Type called “InverseFunctionalProperty” to express this semantics. Furthermore the adhoc query presented in Section 6.22 MUST be available in the Registry to retrieve inverse functional Associations of a ClassificationNode.

4.4 OWL Property Restrictions in ebXML RIM

An 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.

For example, we may define a property "paymentMethod" for travel Web services in general and we may state that the range of this property is the class "PossiblePaymentMethods". Then, for "AirReservationServices", we may wish to restrict "paymentMethod" property to, say, "CreditCard" class as demonstrated in the following two examples:

```
<owl:ObjectProperty rdf:ID="paymentMethod">
  <rdfs:domain rdf:resource="#TravelWebService"/>
  <rdfs:range rdf:resource="#PossiblePaymentMethods"/>
</owl:ObjectProperty >
```

Example owl:ObjectProperty “paymentMethod”

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

Example owl:Restriction on ObjectProperty “paymentMethod”

Obviously, this serves only the purpose of reusing the "paymentMethod" property. Otherwise, a new property "paymentMethodCC" can be defined between "AirReservationServices" and the "CreditCard" classes as shown in the following:

```
<owl:ObjectProperty rdf:ID="paymentMethodCC">
  <rdfs:domain rdf:resource="#AirReservationServices"/>
  <rdfs:range rdf:resource="#CreditCard"/>
</owl:ObjectProperty >
```

Example owl:ObjectProperty “paymentMethodCC”

We believe that defining a generic Association Type and and keeping track of its various restrictions in relational tables will bring considerable overhead to the system. Since an Association Type can always be defined in ebXML between any RergistryObjects, we also think that the expressive power is already there.

4.5 Representing OWL Restricted Cardinality in ebXML RIM

4.5.1 owl:minCardinality (only 0 or 1)

In OWL, cardinality is stated on a property with respect to a particular class. If a minCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at least one individual by that property. This restriction is another way of saying that the property is required to have a value for all instances of the class. In OWL Lite, the only minimum cardinalities allowed are 0 or 1. A minimum cardinality of zero on a property just states (in the absence of any more specific information) that the property is optional with respect to a class [McGuinness, Harmelen].

Consider for example the following OWL code which states that each instance of a “WebService” class must have at least one price:

```
<owl:Class rdf:ID="WebService">
     <rdfs:subClassOf>
          <owl:Restriction>
              <owl:onProperty rdf:resource="#hasPrice"/>
              <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
1 </owl:minCardinality>
          </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Example owl:minCardinality

In ebXML RIM, cardinalities of Association Types MUST be defined by associating a minCardinality slot with the Association Types as shown in the following example:

```
<rim:Association id = "hasPriceMinCardinalityRestriction"
associationType = "urn:oasis:names:tc:ebxml-
regrep:AssociationType:ObjectProperty" sourceObject = "WebService"
targetObject = "Price">
       <rim:Name>
```

```
                <rim:LocalizedString value = 'hasPrice' />
        </rim:Name>
        <rim:Slot name="minCardinality">
                <rim:ValueList>
                        <rim:Value>1</rim:Value>
                </rim:ValueList>
        </rim:Slot>
</rim:Association>
```

Example Representing owl:minCardinality in ebRIM

4.5.2 owl:maxCardinality (only 0 or 1)

In OWL, cardinality is stated on a property with respect to a particular class. If a maxCardinality of 1 is stated on a property with respect to a class, then any instance of that class will be related to at most one individual by that property. A maxCardinality 1 restriction is sometimes called a functional or unique property. It may be useful to state that certain classes have no values for a particular property. This situation is represented by a maximum cardinality of zero on the property [McGuinness, Harmelen].

Consider for example the following OWL code which states that each instance of a “WebService” class can have at most one price:

```
<owl:Class rdf:ID="WebService">
     <rdfs:subClassOf>
          <owl:Restriction>
              <owl:onProperty rdf:resource="#hasPrice"/>
              <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
1 </owl:maxCardinality>
          </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Example owl:maxCardinality

In ebXML RIM, cardinalities of Association Types MUST be defined by associating a maxCardinality slot with the Association Types as shown in the following example:

```
<rim:Association id = "hasPriceMaxCardinalityRestriction"
associationType = "urn:oasis:names:tc:ebxml-
regrep:AssociationType:ObjectProperty" sourceObject = "WebService"
targetObject = "Price">
        <rim:Name>
                <rim:LocalizedString value = 'hasPrice' />
        </rim:Name>
        <rim:Slot name="maxCardinality">
                <rim:ValueList>
                        <rim:Value>1</rim:Value>
                </rim:ValueList>
        </rim:Slot>
</rim:Association>
```

Example Representing owl:maxCardinality in ebRIM

4.5.3 owl:cardinality

In OWL, cardinality is provided as a convenience when it is useful to state that a property on a class has both minCardinality 0 and maxCardinality 0 or both minCardinality 1 and maxCardinality 1 [McGuinness, Harmelen].

Consider for example the following OWL code which states that each instance of a “WebService” class must have exactly one price:

```
<owl:Class rdf:ID="WebService">
     <rdfs:subClassOf>
          <owl:Restriction>
              <owl:onProperty rdf:resource="#hasPrice"/>
```

```
             <owl:Cardinality rdf:datatype="&xsd;nonNegativeInteger"> 1
</owl:Cardinality>
          </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Example owl:Cardinality

In ebXML RIM, cardinalities of Association Types MUST be defined by associating a Cardinality slot with the Association Types as shown in the following example:

```
<rim:Association id = "hasPriceCardinalityRestriction"
associationType = "urn:oasis:names:tc:ebxml-
regrep:AssociationType:ObjectProperty" sourceObject = "WebService"
targetObject = "Price">
       <rim:Name>
             <rim:LocalizedString value = 'hasPrice' />
       </rim:Name>
       <rim:Slot name="cardinality">
             <rim:ValueList>
                    <rim:Value>1</rim:Value>
             </rim:ValueList>
       </rim:Slot>
</rim:Association>
```

Example Representing owl:Cardinality in ebRIM

4.6 Representing OWL Class Intersection in ebXML RIM

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. In the following example, "AirReservationServices" is defined as the intersection of "AirServices" and "ReservationServices":

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

Example owl:intersectionOf

In ebXML RIM "owl:intersectionOf" set operator MUST be represented as follows:

- A new Association Type called "intersectionOf" MUST be created.
- A new ClassificationNode to denote the intersection of the classes MUST be created. For the example, this could be “AirReservationServices” ClassificationNode.
- Each of the intersected classes MUST be represented as members of a new RegistryPackage. For the example, the RegistryPackage should contain “AirServices” and the “RegistrationServices”.
- The new ClassificationNode denoting the intersection MUST be assigned as the sourceObject of the “intersectionOf" association. For the example, “AirReservationServices” must be the the sourceObject of the “intersectionOf" association.
- The target class of the “intersectionOf" association MUST be set to the newly created RegistryPackage. For the example given above, the RegistryPackage containing “AirServices” and the “RegistrationServices” should be the target class of the “intersectionOf" association.

```
<rim:ClassificationNode id = "AirReservationServices" parent= "Service"
code = "AirReservationServices”>
       <rim:Name>
             <rim:LocalizedString value = "AirReservationServices" />
       </rim:Name>
</rim:ClassificationNode>


<rim:RegistryPackage id = "IntersectionOfRegistryPackage" >
             <rim:Name>
                    <rim:LocalizedString value =
"IntersectionOfRegistryPackage"/>
             </rim:Name>
</rim:RegistryPackage>

<rim:Association id = "HasMemberRegistryPackageAssoc1"
associationType = "urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasMember" sourceObject =
"IntersectionOfRegistryPackage"
targetObject = "AirServices" />

<rim:Association id = "HasMemberRegistryPackageAssoc2"
associationType = "urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasMember" sourceObject =
"IntersectionOfRegistryPackage"
targetObject = "ReservationServices" />

<rim:Association id = "IntersectionOfRegistryPackageAssoc"
associationType = "urn:oasis:names:tc:ebxml-
regrep:AssociationType:IntersectionOf" sourceObject =
"AirReservationServices"
targetObject = " IntersectionOfRegistryPackage " />

```

Example Defining Intersection of ClassificationNodes in ebRIM

When such a representation is used to create a complex class (a new ClassificationNode) in RIM, it becomes possible to infer that the objects (instances) classified by both of the classes (ClassificationNodes) constituting the intersection are also the instances of this complex class. The adhoc query presented in Section 6.23 MUST be available in the ebXML Registry to retrieve the direct instances of the complex class and also the instances of the intersection of the classes.

4.7 Representing OWL Versioning in ebXML RIM

4.7.1 owl:versionInfo, owl:priorVersion

An owl:versionInfo statement generally has as its object a string giving information about this version, for example RCS/CVS keywords. This statement does not contribute to the logical meaning of the ontology other than that given by the RDF(S) model theory [McGuinness, Harmelen].

An owl:priorVersion statement contains a reference to another ontology. This identifies the specified ontology as a prior version of the containing ontology [McGuinness, Harmelen].

In ebXML, since a RegistryObject MAY have several versions, a logical id (called lid) is also defined which is unique for different logical objects. However the lid attribute value MUST be the same for all versions of the same logical object. Therefore, almost all the underlying ebXML relational tables keep version information through “versionName” and “comment_” attributes.

“owl:version” information MUST be stored in the “versionName” and “comment_” attributes of the table ClassScheme in the Registry.

4.8 Representing OWL Annotation Properties in ebXML RIM

4.8.1 rdfs:label

rdfs:label is an instance of rdf:Property that may be used to provide a human-readable version of a resource's name [Brickley, Guha].

In ebXML RIM, human readable names of resources are provided through rim:Name. rdfs:label MUST be expressed through rim:Name.

```
<owl:Class rdf:ID="AirReservationServices">
        <rdfs:label>Air Reservation Services</rdfs:label>
</owl:Class>
```

Example rdfs:label

```
<rim:ClassificationNode id = 'AirReservationServices' parent=
'TravelServices' code = 'AirReservationServices'>
        <rim:Name>
               <rim:LocalizedString value = 'Air Reservation Services' />
        </rim:Name>
</rim:ClassificationNode>
```

Example rim:Name

4.8.2 rdfs:comment

rdfs:comment is an instance of rdf:Property that may be used to provide a human-readable description of a resource [Brickley, Guha].

In ebXML RIM, this construct MUST be expressed through rim:Description.

```
<owl:Class rdf:ID="AirReservationServices">
        <rdfs:comment>Open Travel Alliance Air Reservation Services
        </rdfs:comment>
</owl:Class>
```

Example rdfs:comment

```
<rim:ClassificationNode id = 'AirReservationServices' parent=
'TravelServices' code = 'AirReservationServices'>
        <rim:Description>
           <rim:LocalizedString value = 'Open Travel Alliance Air
Reservation Services'/>
        </rim:Description>
</rim:ClassificationNode>
```

Example: rim:Description

4.8.3 rdfs:seeAlso

rdfs:seeAlso is an instance of rdf:Property that is used to indicate a resource that might provide additional information about the subject resource [Brickley, Guha].

This construct MUST be expressed in ebXML RIM by defining an ExternalLink, called, "seeAlsoExternalLink".

```
<owl:Class rdf:ID="AirReservationServices">
        <rdfs:seeAlso rdf:resource="http://www.opentravel.org" />
</owl:Class>
```

Example rdfs:seeAlso

```
<rim:ClassificationNode id = 'AirReservationServices' parent=
'TravelServices' code = 'AirReservationServices'>
</rim:ClassificationNode>

<rim:ExternalLink id = "seeAlsoExternalLink"
        externalURI= "http://www.opentravel.org" >
</rim:ExternalLink>

```

```
<rim:Association id = 'seeAlsoAssociation'
         associationType = 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:ExternallyLinks'
         sourceObject = 'AirReservationServices'
         targetObject = 'seeAlsoExternalLink' />
```

Example rim:seeAlsoExternalLink

4.9 OWL Datatypes in ebXML RIM

OWL allows the use of XML Schema datatypes to describe part of the datatype domain by simply including their URIs within an OWL ontology [McGuinness, Harmelen]. In ebXML, XML Schema datatypes SHOULD be used by providing an external link from the registry.

The following example demonstrates how XML Schema datatype “integer” can be referenced through an ExternalLink called 'integer' and how to define a DatatypeProperty, namely, “hasPrice”, whose target object is the defined to be ExternalLink 'integer':

```
<rim:ExternalLink id = "integer"
                externalURI="http://www.w3.org/2001/XMLSchema#integer" >
      <rim:Name> <rim:LocalizedString value = "XML Schema integer"/>
                </rim:Name>
</rim:ExternalLink>
<rim:Association id = 'hasPrice' associationType = 'urn:oasis:names:tc:ebxml-
               regrep:AssociationType:DatatypeProperty'
   sourceObject = 'AirReservationServices'
   targetObject = 'integer' >
   <rim:Name> <rim:LocalizedString value ="hasPrice"/></rim:Name>
</rim:Association>

```

Example Corresponding ebRIM construct Asssociation

5 Cataloging Service Profile

The ebXML Regitsry provides the ability for a content cataloging service to be configured for any type of content. The cataloging service serves the following purposes:

- Automates the mapping from the source information model (in this case OWL) to ebRIM. This hides the complexity of the mapping from the OWL publisher and eliminates the need for any special UI tools to be provided by the registry implementor for publishing OWL documents.
- Selectively converts content into ebRIM compatible metadata when the content is cataloged after being published. The generated metadata enables the selected content to be used as parameter(s) in content specific parameterized queries.

This section describes the cataloging service for cataloging OWL content.

An OWL document, when published to an ebXML Registry implementing the OWL Profile, MUST be cataloged as specified in this section using a OWL Content Cataloging Service as defined by [ebRS].

5.1 Invocation Control File

The OWL cataloging service MAY optionally support an invocation control file that declaratively specifies the transforms necessary to catalog published OWL documents.

5.2 Input Metadata

The OWL cataloging service MUST be pre-configured to be automatically invoked when the following types of metadata are published, as defined by the [ebRS] specifications.

These are the only types of metadata that MAY describe a OWL document being published:

- An ExtrinsicObject whose ObjectType references the canonical OWL ClassificationNode specified in Section 7. The ExtrinsicObject MUST have an OWL document as its RepositoryItem.
- An ExternalLink whose ObjectType references the canonical OWL ClassificationNode specified in Section 7. In case of ExternalLink the OWL document MUST be resolvable via a URL described by the value of the externalURI attribute of the ExternalLink. Recall that, in the ExternalLink case the OWL document is not be stored in the repository.

```
<rim:ExtrinsicObject id=”urn:acmeinc:ebxml:registry:3.0:owl”>
...
<rim:ExtrinsicObject>
```

Example of ExtrinsicObject Input Metadata

```
<rim:ExternalLink
  id=”urn:acmeinc:ebxml:registry:3.0:owl”
  externalURI=”http://www.acme.com/owl/ebXMLRegistryService.owl”
  >
...
<rim:ExternalLink>
```

Example of ExternalLink Input Metadata

5.3 Input Content

The OWL cataloging service expects an OWL document as its input content. The input content MUST be processed by the OWL cataloging service regardless of whether it is a RepositoryItem for an ExtrinsicObject or whether it is content external to repository that is referenced by an ExternalLink.

5.4 Output Metadata

This section describes the metadata produced by the OWL cataloging service produces as output.

5.4.1 owl:Class → rim:ClassificationNode

The OWL Cataloging service MUST automatically produce a rim:ClassificationNode instance for each owl:class element within the input OWL or its imports, as specified in the owl:Class → rim:ClassificationNode mapping earlier in this document.

5.4.2 rdf:Property → rim:Association Type Property

The OWL Cataloging service MUST automatically produce an rim:Association instance with associationType Property for each rdf:Property element within the input OWL or its imports, as specified in the rdf:Property → rim:Association Type Property mapping earlier in this document.

5.4.3 rdfs:subPropertyOf → rim:Association Type subPropertyOf

The OWL Cataloging service MUST automatically produce an rim:Association instance with associationType subPropertyOf for each rdfs:subPropertyOf element within the input OWL or its imports, as specified in the rdfs:subPropertyOf → rim:Association Type subPropertyOf mapping earlier in this document.

5.4.4 rdfs:subClassOf → rim:Association Type subClassOf

The OWL Cataloging service MUST automatically produce an rim:Association instance with associationType subClassOf for each rdfs:subClassOf element within the input OWL or its imports, as specified in the rdfs:subClassOf → rim:Association Type subClassOf mapping earlier in this document.

5.4.5 owl:Individual → rim:ExtrinsicObject

The OWL Cataloging service MUST automatically produce rim:ExtrinsicObject instances for each owl:Individual element within the input OWL or its imports, as specified in the owl:Individual → rim:ExtrinsicObject mapping earlier in this document.

5.4.6 owl:equivalentClass, owl:equivalentPropery → rim:Association Type EquivalentTo

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType EquivalentTo for each owl:equivalentClass or owl:equivalentProperty element within the input OWL or its imports, as specified in the owl:equivalentClass, owl:equivalantProperty → rim:Association Type EquivalantTo mapping earlier in this document.

5.4.7 owl:sameAs → rim:Association Type sameAs

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType sameAs for each owl:sameAs element within the input OWL or its imports, as specified in the owl:sameAs → rim:Association Type sameAs mapping earlier in this document.

5.4.8 owl:differentFrom → rim:Association Type differentFrom

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType differentFrom for each owl:differentFrom element within the input OWL or its imports, as specified in the owl:differentFrom → rim:Association Type differentFrom mapping earlier in this document.

5.4.9 owl:AllDifferent → rim:RegistryPackage

The OWL Cataloging service MUST automatically produce rim:RegistryPackage instances for each owl:AllDifferent element within the input OWL or its imports, as specified in the owl:AllDifferent →

rim:RegistryPackage mapping earlier in this document.

5.4.10 owl:ObjectProperty → rim:Association Type objectProperty

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType objectProperty for each owl:ObjectProperty element within the input OWL or its imports, as specified in the owl:ObjectProperty → rim:Association Type objectProperty mapping earlier in this document.

5.4.11 owl:DatatypeProperty → rim:Association Type DatatypeProperty

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType datatypeProperty for each owl:DatatypeProperty element within the input OWL or its imports, as specified in the owl:DatatypeProperty → rim:Association Type datatypeProperty mapping earlier in this document.

5.4.12 owl:TransitiveProperty → rim:Association Type transitiveProperty

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType transitiveProperty for each owl:TransitiveProperty element within the input OWL or its imports, as specified in the owl:TransitiveProperty → rim:Association Type transitiveProperty mapping earlier in this document.

5.4.13 owl:inverseOf → rim:Association Type inverseOf

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType inverseOf for each owl:inverseOf element within the input OWL or its imports, as specified in the owl:inverseOf → rim:Association Type inverseOf mapping earlier in this document.

5.4.14 owl:SymmetricProperty→ rim:Association Type SymetricProperty

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType SymetricProperty for each owl:SymetricProperty element within the input OWL or its imports, as specified in the owl:SymetricProperty → rim:Association Type SymetricProperty mapping earlier in this document.

5.4.15 owl:FunctionalProperty→ rim:Association Type FunctionalProperty

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType FunctionalProperty for each owl:FunctionalProperty element within the input OWL or its imports, as specified in the owl:FunctionalProperty → rim:Association Type FunctionalProperty mapping earlier in this document.

5.4.16 owl:InverseFunctionalProperty→ rim:Association Type InverseFunctionalProperty

The OWL Cataloging service MUST automatically produce rim:Association instances with associationType InverseFunctionalProperty for each owl:InverseFunctionalProperty element within the input OWL or its imports, as specified in the owl:InverseFunctionalProperty → rim:Association Type InverseFunctionalProperty mapping earlier in this document.

5.4.17 owl:minCardinality (only 0 or 1)

The OWL Cataloging service MUST automatically add a slot with name minCardinality to the relevant rim:Association instances for each owl:minCardinality element within the input OWL or its imports, as specified in section 4.5.1 where how to represent owl:minCardinality is described.

5.4.18 owl:maxCardinality (only 0 or 1)

The OWL Cataloging service MUST automatically add a slot with name maxCardinality to the relevant rim:Association instances for each owl:maxCardinality element within the input OWL or its imports, as specified in section 4.5.2 where how to represent owl:maxCardinality is described.

5.4.19 owl:cardinality

The OWL Cataloging service MUST automatically add a slot with name cardinality to the relevant rim:Association instances for each owl:cardinality element within the input OWL or its imports, as specified in section 4.5.3 where how to represent owl:cardinality is described.

5.4.20 owl:intersectionOf

The OWL Cataloging service MUST automatically produce a rim:RegistryPackage and a rim:Association instances with type IntersectionOf for each owl:intersectionOf element within the input OWL or its imports, as specified in section 4.6 where how to represent owl:intersectionOf is described.

5.4.21 rdfs:label

The OWL Cataloging service MUST automatically produce a rim:Name instance for each rdfs:label element within the input OWL or its imports, as specified in section 4.8.1 where how to represent rdfs:label is described.

5.4.22 rdfs:comment

The OWL Cataloging service MUST automatically produce a rim:Description instance for each rdfs:comment element within the input OWL or its imports, as specified in section 4.8.2 where how to represent rdfs:comment is described.

5.4.23 rdfs:seeAlso

The OWL Cataloging service MUST automatically produce a rim:ExternalLink and a rim:Association with type ExternallyLinks instances for each rdfs:seeAlso element within the input OWL or its imports, as specified in section 4.8.3 where how to represent rdfs:seeAlso is described.

6 Discovery Profile

The ebXML Regitsry provides the ability for a user defined parameterized queries to be configured for each type of content. The queries may be as complex or simple as the discovery use case requires. The complexity of the parameterized queries may hidden from the registry client by storing them within the ebXML Registry as instances of the AdhocQuery class, and being invoked by simply providing their parameters. Query parameters are often pattern strings that may contain wildcard characters '%' (matches any number of characters) and '_' (matches exactly one character) as described by [ebRS].

An ebXML Registry SHOULD provide a graphical user interface that displays any configured parameterized query as a form which contains an appropriate field for entering each query parameter.

This chapter defines the queries that MUST be support by an ebXML Registry implementing the OWL Profile for processing the semantics provided in the OWL content. An implementation MAY also support additional discovery queries for OWL content, some of which have already identified in this section.

The queries defined in this chapter are parameterized queries stored in the Registry as instances of the AdhocQuery type, in the same manner as any other RegistryObject.

In the subsequent section each query is described simply in terms of its supported parameters that serve as its search criteria. The actual AdhocQuery instances are much more complex in comparison but they are not exposed to the client making the query. Details on these queries are specified canonically in section 7.3 .

Some of the queries that are necessary to process the semantics involved in OWL documents requires SQL recursion mechanism. Since SQL 92, does not support recursion mechanism, those queries are stated to be implemented optionally. Additionally for these types of discovery queries, the “stored procedures” are presented in Section 7.3.

6.1 All SuperProperties Discovery Query

As presented in Section 4.1.3, a new ebXML RIM Association Type called “SubPropertyOf” MUST be defined to represent rdfs:subPropertyOf in ebRIM. Such a semantic enhancement brings the following processing need: given a property, it should be possible to retrieve all of its super properties. This requires a recursion mechanism in SQL queries.

The freebXML implementation allows various relational database products such as Oracle, PostgreSQL and MS SQL Server 2005 to be used as the database. These products have different support for recursion mechanism in SQL Queries.

The AllSuperProperties discovery query MAY be implemented by an ebXML Registry implementing this profile. It allows the discovery of all super properties of a given property instance (Association instance in ebXML terminology) recursively in a property hierarchy (hierarchy of Association Types) in freebXML Registry implementations using MS SQL Server 2005 as the database.

6.1.1 Parameter $propertyName

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of Associations that have associationType of Property.

6.1.2 Example of All SuperProperties Discovery Query

The following example illustrates how to find all the super properties of a given property having a name containing “creditCardPayment” if the query is implemented as an AdHoc Query.

```
<<rs:RequestSlotList>
        <rim:Slot
                name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
                <rim:ValueList>
                        <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllSuperProperties</rim:Value>
```

```
        </rim:ValueList>
    </rim:Slot>
    <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
        <rim:ValueList>
            <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllSuperProperties</rim:Value>
        </rim:ValueList>
    </rim:Slot>
    <rim:Slot name="$propertyName">
        <rim:ValueList>
            <rim:Value>%creditCardPayment%</rim:Value>
        </rim:ValueList>
    </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
    returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
    <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
    </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of All SuperProperties Discovery Query

6.2 Immediate SuperClass Discovery Query

The Immediate SuperClass discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all of the immediate super classes of a given class.

6.2.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.2.2 Example of Immediate SuperClass Discovery Query

The following example illustrates how to find all the immediate super classes of a given class that have a name containing the string “AirReservationServices” .

```
<rs:RequestSlotList>
    <rim:Slot
        name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
        <rim:ValueList>
            <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateSuperClasses</rim:Value>
        </rim:ValueList>
    </rim:Slot>
    <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
        <rim:ValueList>
            <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateSuperClasses</rim:Value>
        </rim:ValueList>
    </rim:Slot>
    <rim:Slot name="$className">
        <rim:ValueList>
            <rim:Value>%AirReservationServices%</rim:Value>
        </rim:ValueList>
    </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
```

```
      returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
      <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
      </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of Immediate SuperClass Discovery Query

6.3 Immediate SubClass Discovery Query

The Immediate SubClass discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all of the immediate subclasses of a given class.

6.3.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNode.

6.3.2 Example of Immediate SubClasss Discovery Query

The following example illustrates how to find all the immediate subclasses of a given class that have a name containing the string “AirServices” .

```
<rs:RequestSlotList>
      <rim:Slot
            name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
            <rim:ValueList>
                  <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateSubClasses</rim:Value>
            </rim:ValueList>
      </rim:Slot>
      <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
            <rim:ValueList>
                  <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateSubClasses</rim:Value>
            </rim:ValueList>
      </rim:Slot>
      <rim:Slot name="$className">
            <rim:ValueList>
                  <rim:Value>%AirServices%</rim:Value>
            </rim:ValueList>
      </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
      returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
      <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
      </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of Immediate SubClass Discovery Query

6.4 All SuperClasses Discovery Query

It should be noted that, given a class, finding its immediate subclasses, super classes is necessary but not sufficient. Given a class, it should be possible to retrieve all of its subclasses, and all of its super classes. This requires a recursion mechanism in SQL queries. The freebXML implementation allows various relational database products such as Oracle, PostgreSQL and MS SQL Server 2005 to be used as the database. These products have different support for recursion mechanisms in SQL Queries.

The All SuperClasses discovery query MAY be implemented by an ebXML Registry implementing this profile. It allows the discovery of all super classes of a given ClassificationNode recursicely in freebXML Registry implementations using MS SQL Server 2005 as the database.

6.4.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNode.

6.4.2 Example of All SuperClassses Discovery Query

The following example illustrates how to find all the super classes of a given class recursively that have a name containing the string “AirReservationServices” if the query is implemented as an Adhoc Query .

```
<rs:RequestSlotList>
       <rim:Slot
              name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
              <rim:ValueList>
                     <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllSuperClasses</rim:Value>
              </rim:ValueList>
       </rim:Slot>
       <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
              <rim:ValueList>
                     <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllSuperClasses</rim:Value>
              </rim:ValueList>
       </rim:Slot>
       <rim:Slot name="$className">
              <rim:ValueList>
                     <rim:Value>%AirReservationServices%</rim:Value>
              </rim:ValueList>
       </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
       returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of All SuperClasses Discovery Query

6.5 All SubClasses Discovery Query

The All SubClasses discovery query MAY be implemented by an ebXML Registry implementing this profile. It allows the discovery of all subclasses of a given ClassificationNode recursively in a freebXML Registry implementations supporting recursion.

6.5.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNode.

6.5.2 Example of All SubClassses Discovery Query

The following example illustrates how to find all the subclasses of a given class recursively that have a name containing the string “AirServices” , if the query is implemented as an Adhoc Query.

```
<rs:RequestSlotList>
        <rim:Slot
                name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
                <rim:ValueList>
                        <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllSubClasses</rim:Value>
                </rim:ValueList>
        </rim:Slot>
        <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
                <rim:ValueList>
                        <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllSubClasses</rim:Value>
                </rim:ValueList>
        </rim:Slot>
        <rim:Slot name="$className">
                <rim:ValueList>
                        <rim:Value>%AirServices%</rim:Value>
                </rim:ValueList>
        </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
        returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
        <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
        </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of All SubClasses Discovery Query

6.6 EquivalentClasses Discovery Query

The EquivalentClasses discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all the equivalent classes of a given ClassificatioNode.

6.6.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.6.2 Example of EquivalentClasses Discovery Query

The following example illustrates how to find all the equivalent classes of a given class that have a name containing the string “AirServices” .

```
<rs:RequestSlotList>
        <rim:Slot
                name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
                <rim:ValueList>
                        <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindEquivalentClasses</rim:Value>
                </rim:ValueList>
        </rim:Slot>
        <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
                <rim:ValueList>
                        <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindEquivalentClasses</rim:Value>
                </rim:ValueList>
```

```
      </rim:Slot>
      <rim:Slot name="$className">
            <rim:ValueList>
                  <rim:Value>%AirServices%</rim:Value>
            </rim:ValueList>
      </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
      returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
      <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
      </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of Equivalent Classes Discovery Query

6.7 EquivalentProperties Discovery Query

The EquivalentProperties discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all the equivalent properties of a given Association that have associationType of Property.

6.7.1 Parameter $propertyName

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of Associations that have associationType of Property

6.7.2 Example of EquivalentProperties Discovery Query

The following example illustrates how to find all the equivalent properties(Association Type) of a given property (Association Type) that have a name containing the string “paymentMethods” .

```
<rs:RequestSlotList>
      <rim:Slot
            name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
            <rim:ValueList>
                  <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindEquivalentProperties</rim:Value>
            </rim:ValueList>
      </rim:Slot>
      <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
            <rim:ValueList>
                  <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindEquivalentProperties</rim:Value>
            </rim:ValueList>
      </rim:Slot>
      <rim:Slot name="$propertyName">
            <rim:ValueList>
                  <rim:Value>%paymentMethods%</rim:Value>
            </rim:ValueList>
      </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
      returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
      <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
      </rim:QueryExpression>
```

```
</rim:AdhocQuery>
```

Example of Equivalent Properties Discovery Query

6.8 SameExtrinsicObjects Discovery Query

The SameExtrinsicObjects discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all the "ExtrinsicObjects" defined to be the same with a given ExtrinsicObject.

6.8.1 Parameter $extrinsicObjectName

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ExtrinsicObjects.

6.8.2 Example of SameExtrinsicObjects Discovery Query

The following example illustrates how to find all the ExtrinsicObjects that are defined to be the same as the ExtrinsicObject that have a name containing the string “MyDocument” .

```

<rs:RequestSlotList>
	<rim:Slot
		name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
		<rim:ValueList>
			<rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindTheSameExtrinsicObjects</rim:Value>
		</rim:ValueList>
	</rim:Slot>
	<rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
		<rim:ValueList>
			<rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindTheSameExtrinsicObjects</rim:Value>
		</rim:ValueList>
	</rim:Slot>
	<rim:Slot name="$extrinsicObjectName">
		<rim:ValueList>
			<rim:Value>%MyDocument%</rim:Value>
		</rim:ValueList>
	</rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
	returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
	<rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
	</rim:QueryExpression>
</rim:AdhocQuery>
```

Example of SameExtrinsicObjects Discovery Query

6.9 DifferentExtrinsicObjects Discovery Query

The DifferentExtrinsicObjects discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all the "ExtrinsicObjects" defined to be the different from a given ExtrinsicObject.

6.9.1 Parameter $extrinsicObjectName

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ExtrinsicObjects.

6.9.2 Example of DifferentExtrinsicObjects Discovery Query

The following example illustrates how to find all the ExtrinsicObjects that are defined to be different from the ExtrinsicObject that have a name containing the string “MyDocument” .

```
<rs:RequestSlotList>
      <rim:Slot
            name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
            <rim:ValueList>
                  <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindDifferentExtrinsicObjects</rim:Value>
            </rim:ValueList>
      </rim:Slot>
      <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
            <rim:ValueList>
                  <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindDifferentExtrinsicObjects</rim:Value>
            </rim:ValueList>
      </rim:Slot>
      <rim:Slot name="$extrinsicObjectName">
            <rim:ValueList>
                  <rim:Value>%MyDocument%</rim:Value>
            </rim:ValueList>
      </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
      returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
      <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
      </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of DifferentExtrinsicObjects Discovery Query

6.10 AllDifferentRegistryObject Discovery Query

The AllDifferentRegistryObjects discovery query MUST be implemented by an ebXML Registry implementing this profile. Given a RegistryObject, it allows the discovery of all the other member "RegistryObjects" of a Registry package that are defined to be the different from each other through a allDifferent slot.

6.10.1 Parameter $registryObjectName

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of RegistryObjects.

6.10.2 Example of AllDifferentRegistryObjects Discovery Query

The following example illustrates how to find all the RegistryObjects that are defined to be different from the RegistryObject that have a name containing the string “MyDocument” .

```

<rs:RequestSlotList>
      <rim:Slot
            name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
            <rim:ValueList>
```

```
                    <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllDifferent</rim:Value>
             </rim:ValueList>
      </rim:Slot>
      <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
             <rim:ValueList>
                    <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllDifferent</rim:Value>
             </rim:ValueList>
      </rim:Slot>
      <rim:Slot name="$registryObjectName">
             <rim:ValueList>
                    <rim:Value>%MyDocument%</rim:Value>
             </rim:ValueList>
      </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
      returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
      <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
      </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of AllDifferentRegistryObjects Discovery Query

6.11 ObjectProperties Discovery Query

The ObjectProperties discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all of the objectProperties of a given classification node.

6.11.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.11.2 Example of ObjectProperties Discovery Query

The following example illustrates how to find all the object properties of a given classification node having a name containing “AirServices” .

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindObjectProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindObjectProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of ObjectProperties Discovery Query

6.12 ImmediateInheritedObjectProperties Discovery Query

The ImmediateInheritedObjectProperties discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all of the objectProperties of a given classification node including the ones inherited from its immediate super classes.

6.12.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.12.2 Example of ImmediateInheritedObjectProperties Discovery Query

The following example illustrates how to find all the object properties of a given classification node having a name containing “AirServices” including the ones inherited from its immediate super classes.

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateInheritedObjectProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateInheritedObjectProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of ImmediateInheritedObjectProperties Discovery Query

6.13 AllInheritedObjectProperties Discovery Query

It should be noted that, given a class, finding the object properties inherited from immediate super classes is necessary but not sufficient. Given a class, it should be possible to retrieve all of the object properties inherited from its super classes. This requires a recursion mechanism in SQL queries. The freebXML implementation allows various relational database products such as Oracle, PostgreSQL and MS SQL Server 2005 to be used as the database. These products have different support for recursion in SQL Queries.

The AllInheritedObjectProperties discovery query MAY be implemented by an ebXML Registry implementing this profile. It allows the discovery of all inherited ObjectProperties recursively of a given ClassificationNode in a ClassificationScheme in freebXML Registry implementations using MS SQL Server 2005 as the database.

6.13.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.13.2 Example of AllInheritedObjectProperties Discovery Query

The following example illustrates how to find all the object properties of a given classification node having a name containing “AirReservationServices” including the ones inherited from all of its super classes recursively, if the query is implemented as an Adhoc Query.

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllInheritedObjectProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-regrep:query:FindAll
InheritedObjectProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of AllInheritedObjectProperties Discovery Query

6.14 DatatypeProperties Discovery Query

The DatatypeProperties discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all of the datatypeProperties of a given classification node.

6.14.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.14.2 Example of DatatypeProperties Discovery Query

The following example illustrates how to find all the datatype properties of a given classification node having a name containing “AirReservationServices” .

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindDatatypeProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindDatatypeProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of DatatypeProperties Discovery Query

6.15 AllInheritedDatatypeProperties Discovery Query

It should be noted that, given a class, finding the datatype properties inherited from immediate super classes is necessary but not sufficient. Given a class, it should be possible to retrieve all of the datatype properties inherited from its super classes. This requires a recursion mechanism in SQL queries. The freebXML implementation allows various relational database products such as Oracle, PostgreSQL and MS SQL Server 2005 to be used as the database. These products have different support for recursion in SQL Queries.

The AllInheritedDatatypeProperties discovery query MAY be implemented by an ebXML Registry implementing this profile. It allows the discovery of all inherited DatatypeProperties recursively of a given ClassificationNode in a ClassificationScheme in freebXML Registry implementations using MS SQL Server 2005 as the database.

6.15.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.15.2 Example of AllInheritedDatatypeProperties Discovery Query

The following example illustrates how to find all the datatype properties of a given classification node having a name containing “AirReservationServices” including the ones inherited from all of its super classes recursively, if the query is implemented as an Adhoc Query.

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllInheritedDatatypeProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindAllInheritedDatatypeProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices %</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of AllInheritedDatatypeProperties Discovery Query

6.16 TransitiveRelationships Discovery Query

To make any use of the transitive property in ebXML registries, coding is necessary to find out the implied information. The TransitiveRelationships discovery query MUST be implemented by an ebXML Registry implementing this profile to handle this semantics.

Given a class which is a source of a transitive property, this discovery query retrieves not only the target objects of a given transitive property, but if the target objects have the same property, it retrieves their target objects too.

6.16.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.16.2 Parameter $propertyName

This parameter's value SHALL specify a string containing a pattern match against the name attribute value of Associations that have associationType of Property

6.16.3 Example of TransitiveRelationships Discovery Query

The following example illustrates how to retrieve all the target objects of the “succeeds” property of the

“AirReservationServices” including the target objects implied by a transitive property relationship.

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindTransitiveRelationships</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindTransitiveRelationships</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$propertyName">
      <rim:ValueList>
         <rim:Value>%succeeds%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of TransitiveRelationships Discovery Query

6.17 TargetObjects Discovery Query

The TargetObjects discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of the targetObjects from the Registry, given a Classification Node (sourceObject) and a property name (Association Type).

6.17.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.17.2 Parameter $propertyName

This parameter's value SHALL specify a string containing a pattern match against the name attribute value of Associations that have associationType of Property.

6.17.3 Example of TargetObjects Discovery Query

The following example illustrates how to retrieve all the target objects of the “paymentMethod” property of the “AirReservationServices”.

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindTargetObjects</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindTargetObjects</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$propertyName">
      <rim:ValueList>
         <rim:Value>%paymentMethod%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of TargetObjects Discovery Query

6.18 TargetObjectsInverseOf Discovery Query

The TargetObjectsInverseOf discovery query MUST be implemented by an ebXML Registry implementing this profile. Given a Classification Node (sourceObject) and a property name (Association Type), this query retrieves the source objects of the properties which are stated to be inverseOf the property name given as a parameter, and considering the Classification Node name as the targetObject of these properties.

6.18.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.18.2 Parameter $propertyName

This parameter's value SHALL specify a string containing a pattern match against the name attribute value of Associations that have associationType of Property.

6.18.3 Example of TargetObjectsInverseOf Discovery Query

The following example illustrates how to retrieve all the source objects of the properties which are stated to the the inverseOf the property “succeeds”, considering the “AirReservationServices” as the target object of these properties.

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindTOinverseOf</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindTOinverseOf</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$propertyName">
      <rim:ValueList>
         <rim:Value>%succeeds%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of TargetObjectsInverseOf Discovery Query

6.19 InverseRanges Discovery Query

The InverseRanges discovery query MUST be implemented by an ebXML Registry implementing this profile to handle this semantics. Given a Classification Node (sourceObject) and a property name (Association Type), this query retrieves not only the target objects of this property, but also the source objects of the properties which are stated to be inverseOf the property name given as a parameter, and considering the Classification Node name as the targetObject of these properties. This query can be thought as the union of the queries presented in Sections 6.17and 6.18.

6.19.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.19.2 Parameter $propertyName

This parameter's value SHALL specify a string containing a pattern match against the name attribute value of Associations that have associationType of Property

6.19.3 Example of InverseRanges Discovery Query

Consider, for example, the "succeeds" property defined in Section 4.3.3. To denote that a certain Web service instance precedes another during execution, 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>
```

Example owl:inverseOf Property

Assume that we want to find all the Web services which can succeed a given Web service. In such a case, we need not only find all the Web services which succeeds this given Web service, that is the target objects of "succeeds" Association instance, but we also need to find all the sourceObjects of the "precedes" Association instance since "precedes" is declared to be the "inverseOf" succeeds Association instance.

The following example illustrates how to retrieve all the services that “succeeds” “AirReservationServices” by also making use of its “preceeds" property.

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindInverseRanges</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindInverseRanges</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$propertyName">
      <rim:ValueList>
         <rim:Value>%succeeds%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of InverseRanges Discovery Query

6.20 SymmetricProperties Discovery Query

The SymmetricProperties discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all of the Symmetric Properties of a given classification node.

6.20.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.20.2 Example of SymmetricProperties Discovery Query

The following example illustrates how to find all the symmetric properties of a given classification node having a name containing “AirReservationServices” .

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindSymmetricProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindSymmetricProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of SymmetricProperties Discovery Query

6.21 FunctionalProperties Discovery Query

The FunctionalProperties discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all of the Functional Properties of a given classification node.

6.21.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.21.2 Example of FunctionalProperties Discovery Query

The following example illustrates how to find all the functional properties of a given classification node having a name containing “AirReservationServices” .

```
<rs:RequestSlotList>
   <rim:Slot
```

```
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindFunctionalProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindFunctionalProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of Functional Properties Discovery Query

6.22 InverseFunctionalProperties Discovery Query

The InverseFunctionalProperties discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all of the Inverse Functional Properties of a given classification node.

6.22.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.22.2 Example of InverseFunctionalProperties Discovery Query

The following example illustrates how to find all the inverse functional properties of a given classification node having a name containing “AirReservationServices” .

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindInverseFunctionalProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindInverseFunctionalProperties</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of InverseFunctional Properties Discovery Query

6.23 Instances Discovery Query

When an intersection definition is used to create a complex class (a new ClassificationNode) in RIM as described in Section 4.6, it becomes possible to infer that the objects (instances) classified by both of the classes (ClassificationNodes) constituting the intersection are also the instances of this complex class.

The Instances discovery query MUST be implemented by an ebXML Registry implementing this profile. It allows the discovery of all of the direct instances of a given classification node and if it is a complex class which is an intersection two classes, it also allows to retrieve the intersection of the instances of both of the classes involved in the intersection definition.

6.23.1 Parameter $className

This parameter's value SHALL specify a string containing a pattern to match against the name attribute value of ClassificationNodes.

6.23.2 Example of Instances Discovery Query

Consider the “AirReservationServices” definition presented in Section 4.6. The following example illustrates how to find all the direct instances of the “AirReservationServices” and also the instances classified by both “AirServices” and also the “ReservationServices”.

```
<rs:RequestSlotList>
   <rim:Slot
      name="urn:oasis:names:tc:ebxml-
regrep:3.0:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindInstances</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rs:AdhocQueryRequest:queryId">
      <rim:ValueList>
         <rim:Value>urn:oasis:names:tc:ebxml-
regrep:query:FindInstances</rim:Value>
      </rim:ValueList>
   </rim:Slot>
   <rim:Slot name="$className">
      <rim:ValueList>
         <rim:Value>%AirReservationServices%</rim:Value>
      </rim:ValueList>
   </rim:Slot>
</rs:RequestSlotList>

<query:ResponseOption returnComposedObjects="true"
```

```
   returnType="LeafClassWithRepositoryItem"/>

<rim:AdhocQuery id="temporaryId">
   <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
   </rim:QueryExpression>
</rim:AdhocQuery>
```

Example of Instances Discovery Query

7 Canonical Metadata Definitions

This chapter specifies the canonical metadata defined by this profile.

7.1 ObjectType Extensions

The following new extensions to the canonical ObjectType ClassificationScheme are described by this profile:

```
    <rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ExtrinsicObject"
lid="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ExtrinsicObject:OWL"  code="OWL"
id="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ExtrinsicObject:OWL">
      <rim:Name>
        <rim:LocalizedString charset="UTF-8" value="label.OWL"/>
      </rim:Name>
    </rim:ClassificationNode>
```

7.2 AssociationType Extensions

The following new extensions to the AssociationType ClassificationScheme are described by this profile:

```
<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:ObjectProperty"
code="ObjectProperty" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:ObjectProperty">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8"
value="ObjectProperty"/>
       </rim:Name>
</rim:ClassificationNode>
<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:Property"
code="Property" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:Property">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8" value="Property"/>
       </rim:Name>
</rim:ClassificationNode>
<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:SubPropertyOf"
code="SubPropertyOf" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubPropertyOf">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8" value="SubPropertyOf"/>
       </rim:Name>
</rim:ClassificationNode>
<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:SubClassOf"
code="SubClassOf" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8" value="SubClassOf"/>
       </rim:Name>
</rim:ClassificationNode>
```

```
<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:IntersectionOf"
code="IntersectionOf" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:IntersectionOf">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8"
value="IntersectionOf"/>
       </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:SameAs"
code="SameAs" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:SameAs">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8" value="SameAs"/>
       </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:DifferentFrom"
code="DifferentFrom" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:DifferentFrom">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8" value="DifferentFrom"/>
       </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:DatatypeProperty"
code="DatatypeProperty" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:DatatypeProperty">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8"
value="DatatypeProperty"/>
       </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:TransitiveProperty"
code="TransitiveProperty" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:TransitiveProperty">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8"
value="TransitiveProperty"/>
       </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:InverseOf"
code="InverseOf" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:InverseOf">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8" value="InverseOf"/>
       </rim:Name>
</rim:ClassificationNode>
```

```
<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:SymmetricProperty"
code="SymmetricProperty" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:SymmetricProperty">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8"
value="SymmetricProperty"/>
       </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-regrep:AssociationType:FunctionalProperty"
code="FunctionalProperty" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:FunctionalProperty">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8"
value="FunctionalProperty"/>
       </rim:Name>
</rim:ClassificationNode>

<rim:ClassificationNode parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
lid="urn:oasis:names:tc:ebxml-
regrep:AssociationType:InverseFunctionalProperty"
code="InverseFunctionalProperty" id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:InverseFunctionalProperty">
       <rim:Name>
              <rim:LocalizedString charset="UTF-8"
value="InverseFunctionalProperty"/>
       </rim:Name>
</rim:ClassificationNode>
```

Extensions to the AssociationType ClassificationScheme

7.3 Canonical Queries

The following new canonical queries are described by this profile. Note that while these queries are complex, the complexity is hidden from clients by exposing only the query parameters to them.

7.3.1 All SuperProperties Discovery Query

Since recursion is not supported by SQL-92, the stored procedure for this query is presented in this section.

```
   CREATE PROCEDURE findAllSuperProperties
       @propertyName varchar(50)
AS
WITH
       Parents(superPropertyID) AS
       (
              SELECT A3.id
              FROM Association A1, Association A2, Association A3, Name_ N
              WHERE A2.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubPropertyOf' AND
                     A1.id = N.parent AND N.value LIKE @propertyName AND
                     A2.sourceObject = A1.id AND A2.targetObject = A3.id
       UNION ALL
              SELECT A.targetObject
              FROM Association A JOIN Parents P
              ON P.superPropertyID = A.sourceObject
              WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubPropertyOf'
       )
SELECT * FROM Parents
```

```
GO
```

Recursive stored procedure for MS SQL Server 2005 retrieving all super properties of a given property (Association)

7.3.2 Immediate SuperClass Discovery Query

```
   <rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateSuperClasses" id="urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateSuperClasses">
       <rim:Name>
              <rim:LocalizedString
value="label.FindImmediateSuperClasses"/>
       </rim:Name>
       <rim:Description>
              <rim:LocalizedString
value="label.FindImmediateSuperClasses.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
              SELECT C2.*
              FROM ClassificationNode C2, Association A, Name_ N,
ClassificationNode C1
              WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf'' AND
              C1.id = N.parent AND
              N.value LIKE ''$className'' AND
              A.sourceObject = C1.id AND
              A.targetObject = C2.id
       </rim:QueryExpression>
</rim:AdhocQuery>

```

The Adhoc Query retrieving immediate super classes of a given classification node

7.3.3 Immediate SubClass Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateSubClasses" id="urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateSubClasses">
       <rim:Name>
              <rim:LocalizedString value="label.FindImmediateSubClasses"/>
       </rim:Name>
       <rim:Description>
              <rim:LocalizedString
value="label.FindImmediateSubClasses.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
              SELECT C2.*
              FROM ClassificationNode C2, Association A, Name_ N,
ClassificationNode C1
              WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf'' AND
              C1.id = N.parent AND
              N.value LIKE ''$className'' AND
              A.sourceObject = C2.id AND
              A.targetObject = C1.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

The Adhoc Query retrieving immediate subclasses of a given classification node

7.3.4 All SuperClasses Discovery Query

Since recursion is not supported by SQL-92, the stored procedure for this query is presented in this section.

```
   CREATE PROCEDURE findAllSuperClasses
       @className varchar(50)
AS
WITH
       Parents(superClassID) AS
       (
             SELECT C2.id
             FROM Association A, Name_ N, ClassificationNode C1, 
ClassificationNode C2
             WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf' AND
                    C1.id = N.parent AND N.value LIKE @className AND
                    A.sourceObject = C1.id AND A.targetObject = C2.id
       UNION ALL
             SELECT A.targetObject
             FROM Association A JOIN Parents P
             ON P.superClassID = A.sourceObject
             WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf'
       )
SELECT * FROM Parents
GO
```

Recursive stored procedure for MS SQL Server 2005 database retrieving all super classes of a given classification node

7.3.5 All SubClasses Discovery Query

Since recursion is not supported by SQL-92, the stored procedure for this query is presented in this section.

```
  CREATE PROCEDURE findAllSubClasses
       @className varchar(50)
AS
WITH
       Children(subClassID) AS
       (
             SELECT C1.id
             FROM Association A, Name_ N, ClassificationNode C1, 
ClassificationNode C2
             WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf' AND
                    C2.id = N.parent AND N.value LIKE @className AND
                    A.sourceObject = C1.id AND A.targetObject = C2.id
       UNION ALL
             SELECT A.sourceObject
             FROM Association A JOIN Children C
             ON C.subClassID = A.targetObject
             WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf'
       )
SELECT * FROM Children
GO
```

Recursive stored procedure for MS SQL Server 2005 database retrieving all subclasses of a given classification node

7.3.6 EquivalentClasses Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindEquivalentClasses" id="urn:oasis:names:tc:ebxml-
regrep:query:FindEquivalentClasses">
       <rim:Name>
             <rim:LocalizedString value="label.FindEquivalentClasses"/>
       </rim:Name>
       <rim:Description>
```

```
             <rim:LocalizedString 
value="label.FindEquivalentClasses.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
             SELECT C2.*
             FROM ClassificationNode C2, Association A, Name_ N, 
ClassificationNode C
             WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:EquivalentTo'' AND
             C.id = N.parent AND
             N.value LIKE ''$className'' AND
             A.sourceObject = C.id AND
             A.targetObject = C2.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the equivalent classes of a given classification node

7.3.7 EquivalentProperties Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindEquivalentProperties" id="urn:oasis:names:tc:ebxml-
regrep:query:FindEquivalentProperties">
       <rim:Name>
             <rim:LocalizedString 
value="label.FindEquivalentProperties"/>
       </rim:Name>
       <rim:Description>
             <rim:LocalizedString 
value="label.FindEquivalentProperties.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
             SELECT A3.*
             FROM Association A3, Association A1, Name_ N, Association 
A2
             WHERE A1.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:EquivalentTo'' AND
             A2.id = N.parent AND
             N.value LIKE ''$propertyName'' AND
             A1.sourceObject = A2.id AND
             A1.targetObject = A3.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the equivalent Association Type of a given Association Type

7.3.8 SameExtrinsicObjects Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindTheSameExtrinsicObjects" id="urn:oasis:names:tc:ebxml-
regrep:query:FindTheSameExtrinsicObjects">
       <rim:Name>
             <rim:LocalizedString 
value="label.FindTheSameExtrinsicObjects"/>
       </rim:Name>
       <rim:Description>
             <rim:LocalizedString 
value="label.FindTheSameExtrinsicObjects.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
             SELECT E2.*
             FROM ExtrinsicObject E2, Association A, Name_ N, 
ExtrinsicObject E
```

```
                WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:SameAs'' AND
                E.id = N.parent AND
                N.value LIKE ''$extrinsicObjectName'' AND
                A.sourceObject = E.id AND
                A.targetObject = E2.id
        </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the "ExtrinsicObjects" defined to be the same with a given ExtrinsicObject

7.3.9 DifferentExtrinsicObjects Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindDifferentExtrinsicObjects" id="urn:oasis:names:tc:ebxml-
regrep:query:FindDifferentExtrinsicObjects">
        <rim:Name>
                <rim:LocalizedString
value="label.FindDifferentExtrinsicObjects"/>
        </rim:Name>
        <rim:Description>
                <rim:LocalizedString
value="label.FindDifferentExtrinsicObjects.desc"/>
        </rim:Description>
        <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
                SELECT E2.*
                FROM ExtrinsicObject E2, Association A, Name_ N,
ExtrinsicObject E
                WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:DifferentFrom'' AND
                E.id = N.parent AND
                N.value LIKE ''$extrinsicObjectName'' AND
                A.sourceObject = E.id AND
                A.targetObject = E2.id
        </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the "ExtrinsicObjects" defined to be different from a given ExtrinsicObject

7.3.10 AllDifferentRegistryObject Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindAllDifferent" id="urn:oasis:names:tc:ebxml-
regrep:query:FindAllDifferent">
        <rim:Name>
                <rim:LocalizedString value="label.FindAllDifferent"/>
        </rim:Name>
        <rim:Description>
                <rim:LocalizedString value="label.FindAllDifferent.desc"/>
        </rim:Description>
        <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
                SELECT RO2.*
                FROM RegistryObject RO2, Association A1, Association A2,
Name_ N, RegistryObject RO,
                RegistryPackage RP<!--, Slot S-->
                WHERE A1.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasMember'' AND
                RO.id = N.parent AND
                N.value LIKE ''$registryObjectName'' AND
                A1.sourceObject = RP.id AND
                <!-- S.parent = RP.id AND
                S.name_ LIKE ''allDifferent'' AND S.value LIKE ''true'' AND
-->
```

```
              A1.targetObject = RO.id AND
              A2.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasMember'' AND
              A2.sourceObject = RP.id AND
              A2.targetObject != RO.id AND
              A2.targetObject = RO2.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the "RegistryObjects" defined to be different from a given RegistryObject through a “allDifferentFrom” construct

7.3.11 ObjectProperties Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindObjectProperties" id="urn:oasis:names:tc:ebxml-
regrep:query:FindObjectProperties">
       <rim:Name>
              <rim:LocalizedString value="label.FindObjectProperties"/>
       </rim:Name>
       <rim:Description>
              <rim:LocalizedString
value="label.FindObjectProperties.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
              SELECT A.*
              FROM Association A, Name_ N, ClassificationNode C
              WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:ObjectProperty'' AND
              C.id = N.parent AND
              N.value LIKE ''$className'' AND
              A.sourceObject = C.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the object properties of a given classification node

7.3.12 ImmediateInheritedObjectProperties Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateInheritedObjectProperties"
id="urn:oasis:names:tc:ebxml-
regrep:query:FindImmediateInheritedObjectProperties">
       <rim:Name>
              <rim:LocalizedString
value="label.FindImmediateInheritedObjectProperties"/>
       </rim:Name>
       <rim:Description>
              <rim:LocalizedString
value="label.FindImmediateInheritedObjectProperties.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
              SELECT A2.*
              FROM Association A, Name_ N, ClassificationNode C1,
ClassificationNode C2, Association A2
              WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf'' AND
              C1.id = N.parent AND
              N.value LIKE ''$className'' AND
              A.sourceObject = C1.id AND
              A.targetObject = C2.id AND
              A2.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:ObjectProperty'' AND
              A2.sourceObject=C2.id
       </rim:QueryExpression>
```

```
</rim:AdhocQuery>
```

Adhoc Query retrieving all of the properties of a given classification node including the ones inherited from its immediate super classes

7.3.13 AllInheritedObjectProperties Discovery Query

Since recursion is not supported by SQL-92, the stored procedure for this query is presented in this section.

```
CREATE PROCEDURE findAllInheritedObjectProperties
       @className varchar(50)
AS
WITH Parents(superClassID) AS (
        SELECT C2.id
        FROM Association A, Name_ N, ClassificationNode C1, 
ClassificationNode C2
        WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf' AND
               C1.id = N.parent AND N.value LIKE @className AND
               A.sourceObject = C1.id AND A.targetObject = C2.id
 UNION ALL
        SELECT A.targetObject
        FROM Association A JOIN Parents P
        ON P.superClassID = A.sourceObject
        WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf'
) SELECT A.id
  FROM Association A, Parents P
  WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:ObjectProperty' AND
       A.sourceObject=P.superClassID
UNION

SELECT A.id
FROM Name_ N, ClassificationNode C, Association A
WHERE C.id = N.parent AND N.value LIKE @className AND
      A.sourceObject=C.id AND A.associationType LIKE
'urn:oasis:names:tc:ebxml-regrep:AssociationType:ObjectProperty'
GO
```

Recursive stored procedure for MS SQL Server 2005 database retrieving all inherited Object Properties of a given classification node

7.3.14 DatatypeProperties Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindDatatypeProperties" id="urn:oasis:names:tc:ebxml-
regrep:query:FindDatatypeProperties">
       <rim:Name>
              <rim:LocalizedString value="label.FindDatatypeProperties"/>
       </rim:Name>
       <rim:Description>
              <rim:LocalizedString 
value="label.FindDatatypeProperties.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
              SELECT A.*
              FROM Association A, Name_ N, ClassificationNode C
              WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:DatatypeProperty'' AND
              C.id = N.parent AND
              N.value LIKE ''$className'' AND
              A.sourceObject = C.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the datatype properties of a given classification node

7.3.15 AllInheritedDatatypeProperties Discovery Query

Since recursion is not supported by SQL-92, the stored procedure for this query is presented in this section.

```
CREATE PROCEDURE findAllInheritedDatatypeProperties
       @className varchar(50)
AS
WITH Parents(superClassID) AS (
        SELECT C2.id
        FROM Association A, Name_ N, ClassificationNode C1, 
ClassificationNode C2
        WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf' AND
               C1.id = N.parent AND N.value LIKE @className AND
               A.sourceObject = C1.id AND A.targetObject = C2.id
 UNION ALL
        SELECT A.targetObject
        FROM Association A JOIN Parents P
        ON P.superClassID = A.sourceObject
        WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:SubClassOf'
) SELECT A.id
  FROM Association A, Parents P
  WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
regrep:AssociationType:DatatypeProperty' AND
       A.sourceObject=P.superClassID
UNION

SELECT A.id
FROM Name_ N, ClassificationNode C, Association A
WHERE C.id = N.parent AND N.value LIKE @className AND
      A.sourceObject=C.id AND A.associationType LIKE 
'urn:oasis:names:tc:ebxml-regrep:AssociationType:DatatypeProperty'
GO
```

Recursive stored procedure for MS SQL Server 2005 database retrieving all inherited Datatype Properties of a given classification node

7.3.16 TransitiveRelationships Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindTransitiveRelationships" id="urn:oasis:names:tc:ebxml-
regrep:query:FindTransitiveRelationships">
       <rim:Name>
             <rim:LocalizedString 
value="label.FindTransitiveRelationships"/>
       </rim:Name>
       <rim:Description>
             <rim:LocalizedString 
value="label.FindTransitiveRelationships.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
             SELECT C.*
             FROM ClassificationNode C, Association A1, Association A2, 
Name_ N1, Name_ N2, Name_ N3
             WHERE A1.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType: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 ''$propertyName'' AND
              A2.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:TransitiveProperty'' AND
              A2.targetObject = C.id
              <!-- UNION
              SELECT C.*
              FROM ClassificationNode C, Association A1, Name_ N1, Name_
N3
              WHERE A1.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:TransitiveProperty'' AND
              A1.id = N1.parent AND
              N1.value LIKE ''$propertyName'' AND
              A1.sourceObject = N3.parent AND
              N3.value LIKE ''$className'' AND
              A1.targetObject = C.id -->
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving the objects in transitive relationship with a given object

7.3.17 TargetObjects Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindTargetObjects" id="urn:oasis:names:tc:ebxml-
regrep:query:FindTargetObjects">
       <rim:Name>
              <rim:LocalizedString value="label.FindTargetObjects"/>
       </rim:Name>
       <rim:Description>
              <rim:LocalizedString value="label.FindTargetObjects.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
              SELECT C2.*
              FROM ClassificationNode C2, Association A, Name_ N, Name_
N2, ClassificationNode C1
              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
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving the Target Objects from the Registry, given a Source Object and an Association

7.3.18 TargetObjectsInverseOf Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindTOinverseOf" id="urn:oasis:names:tc:ebxml-
regrep:query:FindTOinverseOf">
       <rim:Name>
              <rim:LocalizedString value="label.FindTOinverseOf"/>
       </rim:Name>
       <rim:Description>
              <rim:LocalizedString value="label.FindTOinverseOf.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
              SELECT C2.*
              FROM ClassificationNode C2, Association A1, Association A2,
Association A3, Name_ N, Name_ N2, ClassificationNode C1
              WHERE A2.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:InverseOf'' AND
              A1.id = N.parent AND
```

```
             N.value LIKE ''$propertyName'' AND
             A2.sourceObject = A1.id AND
             A2.targetObject = A3.id AND
             C1.id = N2.parent AND
             N2.value LIKE ''$className'' AND
             A3.targetObject = C1.id AND
             A3.sourceObject = C2.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc query retrieving the Source Objects of an Association which is in "inverseOf" relationship to this Association

7.3.19 InverseRanges Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindInverseRanges" id="urn:oasis:names:tc:ebxml-
regrep:query:FindInverseRanges">
       <rim:Name>
             <rim:LocalizedString value="label.FindInverseRanges"/>
       </rim:Name>
       <rim:Description>
             <rim:LocalizedString value="label.FindInverseRanges.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
             <!-- SELECT C2.*
             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 C2.*
             FROM ClassificationNode C2, Association A1, Association A2,
Association A3, Name_ N, NAME_ N2, ClassificationNode C1
             WHERE A2.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:InverseOf'' AND
             A1.id = N.parent AND
             N.value LIKE ''$propertyName'' AND
             A2.sourceObject = A1.id AND
             A2.targetObject = A3.id AND
             C1.id = N2.parent AND
             N2.value LIKE ''$className'' AND
             A1.sourceObject = C1.id AND
             A3.sourceObject = C2.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query Retrieving both the Target Objects of a given Association and the Source Objects of an Association which is in "inverseOf" relationship to this Association

7.3.20 SymmetricProperties Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindSymmetricProperties" id="urn:oasis:names:tc:ebxml-
regrep:query:FindSymmetricProperties">
       <rim:Name>
             <rim:LocalizedString value="label.FindSymmetricProperties"/>
       </rim:Name>
       <rim:Description>
             <rim:LocalizedString
value="label.FindSymmetricProperties.desc"/>
       </rim:Description>
      <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
             SELECT A.*
             FROM Association A, Name_ N, ClassificationNode C
             WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:SymmetricProperty'' AND
             C.id = N.parent AND
             N.value LIKE ''$className'' AND
             A.sourceObject = C.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the Symmetric properties of a given classification node

7.3.21 FunctionalProperties Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindFunctionalProperties" id="urn:oasis:names:tc:ebxml-
regrep:query:FindFunctionalProperties">
       <rim:Name>
              <rim:LocalizedString
value="label.FindFunctionalProperties"/>
       </rim:Name>
       <rim:Description>
              <rim:LocalizedString
value="label.FindFunctionalProperties.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
              SELECT A.*
              FROM Association A, Name_ N, ClassificationNode C
              WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:FunctionalProperty'' AND
              C.id = N.parent AND
              N.value LIKE ''$className'' AND
              A.sourceObject = C.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the Functional properties of a given classification node

7.3.22 InverseFunctionalProperties Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-
regrep:query:FindInverseFunctionalProperties"
id="urn:oasis:names:tc:ebxml-
regrep:query:FindInverseFunctionalProperties">
       <rim:Name>
              <rim:LocalizedString
value="label.FindInverseFunctionalProperties"/>
       </rim:Name>
       <rim:Description>
              <rim:LocalizedString
value="label.FindInverseFunctionalProperties.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
              SELECT A.*
              FROM Association A, Name_ N, ClassificationNode C
              WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:InverseFunctionalProperty'' AND
              C.id = N.parent AND
              N.value LIKE ''$className'' AND
              A.sourceObject = C.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query retrieving all the Inverse Functional properties of a given classification node

7.3.23 Instances Discovery Query Discovery Query

```
<rim:AdhocQuery lid="urn:oasis:names:tc:ebxml-regrep:query:FindInstances"
id="urn:oasis:names:tc:ebxml-regrep:query:FindInstances">
       <rim:Name>
             <rim:LocalizedString value="label.FindInstances"/>
       </rim:Name>
       <rim:Description>
             <rim:LocalizedString value="label.FindInstances.desc"/>
       </rim:Description>
       <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-
regrep:QueryLanguage:SQL-92">
             <!-- SELECT S.* FROM Service S, (
             SELECT A.targetObject AS id
             FROM RegistryPackage R, Association A
             WHERE R.id=A.sourceObject AND
             A.associationType = ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasMember'' AND
             R.id IN (
             SELECT A.targetObject
             FROM Association A, Name_ N, ClassificationNode C
             WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:IntersectionOf'' AND
             C.id = N.parent AND
             N.value LIKE ''$className'' AND
             A.sourceObject = C.id
             )
             ) AS T1, (
             SELECT A.targetObject AS id
             FROM RegistryPackage R, Association A
             WHERE R.id=A.sourceObject AND
             A.associationType = ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasMember'' AND
             R.id IN (
             SELECT A.targetObject
             FROM Association A, Name_ N, ClassificationNode C
             WHERE A.associationType LIKE ''urn:oasis:names:tc:ebxml-
regrep:AssociationType:IntersectionOf'' AND
             C.id = N.parent AND
             N.value LIKE ''$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
             UNION -->
             SELECT S.*
             FROM Service S, Classification C, ClassificationNode CN,
Name_ N
             WHERE S.id = C. classifiedObject AND
             C.classificationNode = CN.id AND
             N.value LIKE ''$className'' AND
             N.parent = CN.id
       </rim:QueryExpression>
</rim:AdhocQuery>
```

Adhoc Query Retrieving the instances of intersected classes

8 OWL Profile References

8.1 Normative References

[Brickley, Guha] Brickley, D., Guha, R.V., RDF Vocabulary Description Language 1.0: RDF Schema
W3C Recommendation 10 February 2004
http://www.w3.org/TR/rdf-schema/

[DAML+OIL] http://www.daml.org/

[ebRIM] ebXML Registry Information Model version 3.0
http://docs.oasis-open.org/regrep/regrep-rim/v3.0/regrep-rim-3.0-os.pdf

[ebRS] ebXML Registry Services Specification version 3.0
http://docs.oasis-open.org/regrep/regrep-rs/v3.0/regrep-rs-3.0-os.pdf

[UML] Unified Modeling Language version 1.5
http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.pdf

[ebRR-DPT] Deployment Profile Template For ebXML Registry 3.0 OASIS Specifications V_0.1.2

[ebMS-DPT] Deployment Profile Template For OASIS Specification ebXML Message Service 2.0

[OWL] Web Ontology Language (OWL), http://www.w3.org/2004/OWL/

[RDF] Resource Description Framework, http://www.w3.org/RDF/

[SQL 92] SQL ISO/IEC 9075:1992 Information technology - Database languages - SQL.

[SQL 99] ISO/IEC 9075:1999(E) Information technology - Database languages - SQL.

[WSDL] WSDL Specification
http://www.w3.org/TR/wsdl

8.2 Informative References

[Dogac, et. al.] Dogac A., Kabak Y., Laleci G. C. Mattocks, F. Najmi, J. Pollock
Enhancing ebXML Registries to Make them OWL Aware
Distributed and Parallel Databases Journal, Springer-Verlag, Vol. 18, No. 1, July 2005, pp. 9-36.

[IMPL] ebXML Registry 3.0 Implementations
freebXML Registry: A royalty free, open source ebXML Registry Implementation
http://ebxmlrr.sourceforge.net

[LeeHendler]
Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web", Scientific American, May 2001.

[McGuinness, Harmelen] OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-features/

[StaabStuder] Staab, S., Studer, R., Handbook on Ontologies, Springer, 2004.