



ebXML Registry Profile for Web Ontology

Language (OWL)

Version 0.1 Draft 1

Draft OASIS Profile, January 11, 2006

Document identifier:

Jan16-2006regrep-owl-profile-1.0

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.

19

20 **Status:**

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

25 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.

29 For information on whether any patents have been disclosed that may be essential to
30 implementing this specification, and any offers of patent licensing terms, please refer to the
31 Intellectual Property Rights section of the OASIS ebXML Registry TC web page (<http://www.oasis-open.org/committees/regrep/>).
32

1 Table of Contents

33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77

1 Table of Contents.....	3
1 Introduction.....	7
1.1 Terminology.....	7
1.2 Conventions.....	8
2 OWL Overview.....	9
2.1 RDF Schema Features.....	9
2.2 (In)Equality.....	9
2.3 Property Characteristics	9
2.4 Property Restrictions.....	10
2.5 Restricted Cardinality.....	10
2.6 Class Intersection.....	10
2.7 Versioning.....	10
2.8 Annotation Properties	10
2.9 Datatypes	10
3 ebXML Registry Overview.....	11
3.1 Overview of [ebRIM].....	11
3.1.1 RegistryObject.....	12
3.1.2 Object Identification.....	12
3.1.3 Object Naming and Description.....	13
3.1.4 Object Attributes.....	13
3.1.4.1 Slot Attributes.....	13
3.1.5 Object Classification.....	14
3.1.6 Object Association.....	14
3.1.7 Object References To Web Content.....	15
3.1.8 Object Packaging.....	15
3.1.9 ExtrinsicObject	16
3.1.10 Service Description.....	16
3.2 Overview of [ebRS].....	16
4 Representing OWL Constructs in ebRIM and Providing Processing Support for Additional Semantics..	17
4.1 Representing RDF Schema Features in ebRIM.....	17
4.1.1 owl:Class → rim:ClassificationNode.....	17
4.1.2 rdf:Property → rim:Association Type Property.....	17
4.1.3 rdfs:subPropertyOf → rim:Association Type subPropertyOf.....	18
4.1.4 rdfs:subClassOf → rim:Association Type subClassOf.....	19
4.1.5 owl:Individual → rim:ExtrinsicObject.....	22
4.2 Representing OWL (In)Equality Constructs in ebXML RIM.....	23
4.2.1 owl:equivalentClass, owl:equivalentProperty → rim:Association Type EquivalentTo	23
4.2.2 owl:sameAs → rim:Association Type sameAs.....	24
4.2.3 owl:differentFrom → rim:Association Type differentFrom.....	24
4.2.4 owl:AllDifferent.....	25
4.3 Representing OWL Property Characteristics in ebRIM.....	26
4.3.1 owl:ObjectProperty → rim:Association Type objectProperty.....	26
4.3.2 owl:DatatypeProperty → rim:Association Type DatatypeProperty.....	29
4.3.3 owl:TransitiveProperty → rim:Association Type transitiveProperty.....	30

78	4.3.4 owl:inverseOf → rim:Association Type inverseOf.....	32
79	4.3.4.1 Retrieving the Target Objects of a given Association of a given ClassificationNode.....	32
80	4.3.4.2 Retrieving the Target Objects from the Registry which are in "inverseOf" relationship to a given	
81	Association of a given ClassificationNode.....	33
82	4.3.4.3 A Clarifying Example.....	33
83	4.3.5 owl:SymmetricProperty→ rim:Association Type SymmetricProperty.....	34
84	4.3.6 owl:FunctionalProperty→ rim:Association Type FunctionalProperty.....	35
85	4.3.7 owl:InverseFunctionalProperty→ rim:Association Type InverseFunctionalProperty.....	36
86	4.4 OWL Property Restrictions in ebXML RIM.....	37
87	4.5 Representing OWL Restricted Cardinality in ebXML RIM.....	38
88	4.5.1 owl:minCardinality (only 0 or 1).....	38
89	4.5.2 owl:maxCardinality (only 0 or 1).....	38
90	4.5.3 owl:cardinality.....	39
91	4.6 Representing OWL Class Intersection in ebXML RIM.....	39
92	4.7 Representing OWL Versioning in ebXML RIM.....	42
93	4.7.1 owl:versionInfo, owl:priorVersion.....	42
94	4.8 Representing OWL Annotation Properties in ebXML RIM.....	42
95	4.8.1 rdfs:label.....	42
96	4.8.2 rdfs:comment.....	43
97	4.8.3 rdfs:seeAlso.....	43
98	4.9 OWL Datatypes in ebXML RIM.....	43
99	5 OWL Profile References.....	45
100	5.1 Normative References.....	45
101	5.2 Informative References.....	45
102		

Illustration Index

Figure 1: ebXML Registry Information Model, High Level Public View.....	12
Figure 2: ebXML Registry Information Model, Inheritance View.....	13

Index of Tables

104

1 Introduction

105

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

107 The ebXML Registry holds the metadata for the RegistryObjects and the documents pointed at by the
108 RegistryObjects reside in an ebXML repository. The basic semantic mechanisms of ebXML Registry are
109 classification hierarchies (ClassificationScheme) consisting of ClassificationNodes and the predefined
110 Association Types among RegistryObjects. Furthermore, RegistryObjects can be assigned properties
111 through a slot mechanism and RegistryObjects can be associated with ClassificationNodes through
112 Classification instances. Given these constructs, considerable amount of semantics can be defined in the
113 registry.

114 However, currently semantics is becoming a much broader issue than it used to be since several
115 application domains are making use of ontologies to add the knowledge to their data and applications
116 [StaabStuder]. One of the driving forces for ontologies is the Semantic Web initiative [LeeHendler]. As a
117 part of this initiative, W3C's Web Ontology Working Group defined Web Ontology Language [OWL].

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

120 This document normatively defines the ebXML Registry profile for Web Ontology Language (OWL) Lite.
121 More specifically, this document normatively specifies how OWL Lite constructs SHOULD be represented
122 by ebXML RIM constructs **without causing any changes in the ebXML Registry architecture**
123 **specification**. Furthermore, this document normatively specifies the code to process some of the OWL
124 semantics through parameterized (generic) stored procedures that SHOULD be made available from the
125 ebXML Registry.

126 These predefined stored queries provide the necessary means to exploit the enhanced semantics stored
127 in the Registry. Hence, an application program does not have to develop additional code to process this
128 semantics. In this way, it becomes possible to retrieve not only explicit but also the implied knowledge
129 through queries, the enhancements to the registry are generic and also the registry specification is kept
130 intact. The capabilities provided, move the semantics support beyond what is currently available in ebXML
131 registries and it does so by using a standard ontology language.

132 Finally it worths noting that ontologies can play two major roles: One is to provide a source of shared and
133 precisely defined terms which can be used formalizing knowledge and relationship among objects in a
134 domain of interest. The other is to reason about the ontologies. When an ontology language like OWL is
135 mapped to a class hierarchy like the one in ebXML, the first role can directly be achieved. Furthermore
136 some implicit information can be obtained by predefined parameterized queries. However, when we want
137 full reasoning power, we need reasoners. Yet, OWL reasoners can not directly run on the ebXML registry
138 because all the registry information is stored in OWL syntax.

139 The document is organized as follows:

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

1.1 Terminology

148

149 The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT,
150 RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in IETF RFC
151 2119 [RFC2119].

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

155 1.2 Conventions

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

- 158 • UML Diagrams

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

- 161 • Identifier Placeholders

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

167

```
168 <id="{CANONICAL_OBJECT_TYPE_ID_ORGANIZATION}" >
```

169

2 OWL Overview

170

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

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

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

185 Within the scope of this document, only OWL Lite constructs are considered and in the rest of the
186 document, “OWL” is used to mean “OWL Lite” unless otherwise stated.

187 OWL describes the structure of a domain in terms of classes and properties. In OWL, properties can have
188 multiple domains and multiple ranges. Multiple domain (range) expressions restrict the domain (range) of
189 a property to the intersection of the class expressions.

190

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

2.1 RDF Schema Features

192

- 193 • Class (Thing, Nothing)
- 194 • rdfs:subClassOf
- 195 • rdf:Property
- 196 • rdfs:subPropertyOf
- 197 • rdfs:domain
- 198 • rdfs:range
- 199 • Individual

2.2 (In)Equality

200

- 201 • equivalentClass
- 202 • equivalentProperty
- 203 • sameAs
- 204 • differentFrom
- 205 • AllDifferent
- 206 • distinctMembers

2.3 Property Characteristics

207

- 208 • ObjectProperty
- 209 • DatatypeProperty
- 210 • inverseOf

- 211 • TransitiveProperty
- 212 • SymmetricProperty
- 213 • FunctionalProperty
- 214 • InverseFunctionalProperty

215 2.4 Property Restrictions

- 216 • Restriction
- 217 • onProperty
- 218 • allValuesFrom
- 219 • someValuesFrom

220 2.5 Restricted Cardinality

- 221 • minCardinality (only 0 or 1)
- 222 • maxCardinality (only 0 or 1)
- 223 • cardinality (only 0 or 1)

224 2.6 Class Intersection

- 225 • intersectionOf

226 2.7 Versioning

- 227 • versionInfo
- 228 • priorVersion
- 229 • backwardCompatibleWith
- 230 • incompatibleWith
- 231 • DeprecatedClass
- 232 • DeprecatedProperty

233 2.8 Annotation Properties

- 234 • rdfs:label
- 235 • rdfs:comment
- 236 • rdfs:seeAlso
- 237 • rdfs:isDefinedBy
- 238 • AnnotationProperty
- 239 • OntologyProperty

240 2.9 Datatypes

- 241 • xsd datatypes

242

3 ebXML Registry Overview

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

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

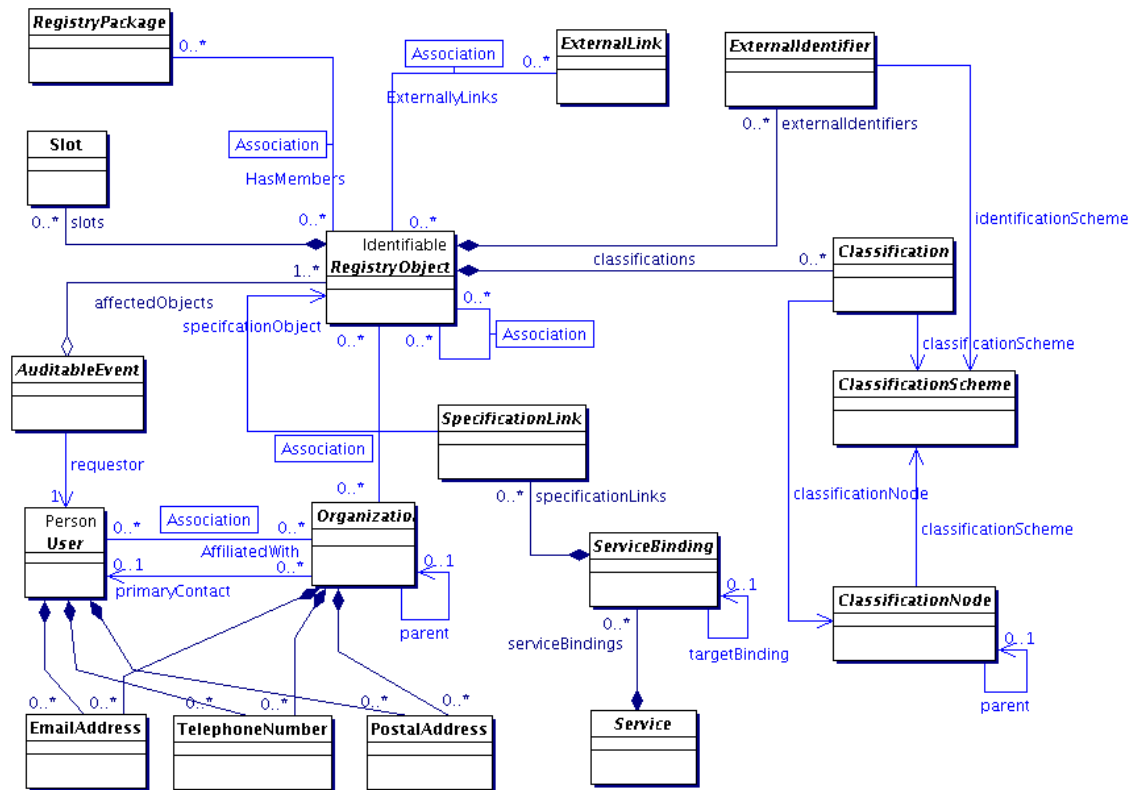
246 The information presented is informative and is not intended to replace the normative information defined
247 by ebXML Registry.

3.1 Overview of [ebRIM]

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

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

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



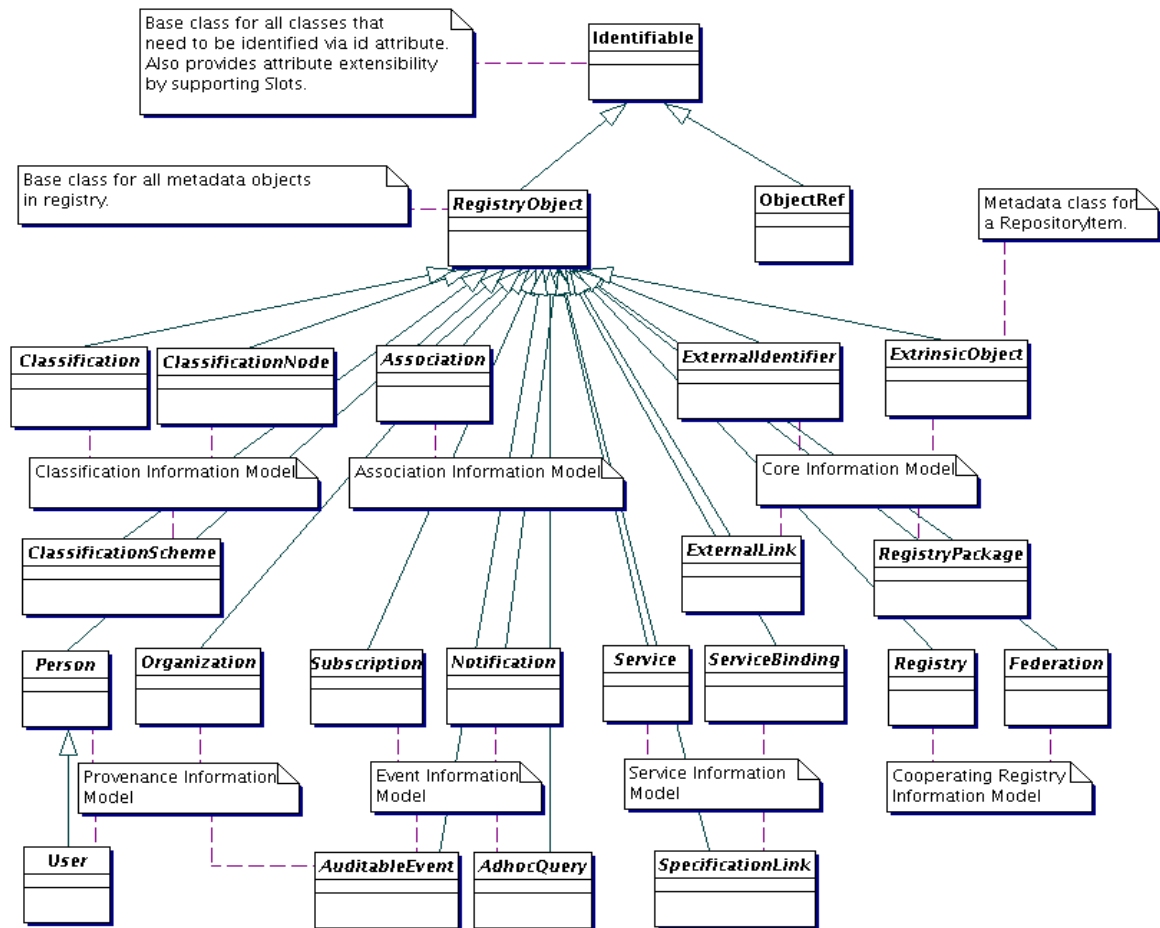
256

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

257

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

262



264 **Figure 2: ebXML Registry Information Model, Inheritance View**

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

266 3.1.1 RegistryObject

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

270 3.1.2 Object Identification

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

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

274 **Listing 1: Example of id attribute**

275 The id attribute value MAY potentially be human friendly.

```
276 <rim:Organization id="urn:oasis:Organization">
```

278 **Listing 2: Example of human friendly id attribute**

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

282 friendly:

283

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

286 **Listing 3: Example of lid Attribute**

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

289

```
290 <rim:Organization id=${ACME_ORG_ID}
291     lid="urn:acme:ACMEOrganization">
292
293     <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
294         identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
295         value="ACME"/>
296     </rim:ExternalIdentifier>
297
298 </rim:Organization>
```

299 **Listing 4: Example of ExternalIdentifier**

300 3.1.3 Object Naming and Description

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

303

```
304 <rim:Organization id=${ACME_ORG_ID}
305     lid="urn:acme:ACMEOrganization">
306
307     <rim:Name>
308         <rim:LocalizedString value="ACME Inc." xml:lang="en-US"/>
309     </rim:Name>
310     <rim:Description>
311         <rim:LocalizedString value="ACME is a provider of Java software."
312             xml:lang="en-US"/>
313     </rim:Description>
314
315     <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
316         identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
317         value="ACME"/>
318     </rim:ExternalIdentifier>
319 </rim:Organization>
```

320 **Listing 5: Example of Name and Description**

321

322 3.1.4 Object Attributes

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

325 3.1.4.1 Slot Attributes

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

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

332

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

335

```

336 <rim:Slot name="urn:acme:slot:NASDAQSymbol">
337   <rim:ValueList>
338     <rim:Value>ACME</rim:Value>
339   </rim:ValueList>
340 </rim:Slot>
341
342   <rim:Name>
343     <rim:LocalizedString value="ACME Inc." xml:lang="en-US"/>
344   </rim:Name>
345   <rim:Description>
346     <rim:LocalizedString value="ACME makes Java. Provider of free Java
347 software."
348       xml:lang="en-US"/>
349   </rim:Description>
350   <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
351     identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
352     value="ACME"/>
353   </rim:ExternalIdentifier>
354 </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.

```

362 <rim:Organization id=${ACME_ORG_ID}
363   lid="urn:acme:ACMEOrganization">
364   <rim:Slot name="urn:acme:slot:NASDAQSymbol">
365     <rim:ValueList>
366       <rim:Value>ACME</rim:Value>
367     </rim:ValueList>
368   </rim:Slot>
369   <rim:Name>
370     <rim:LocalizedString value="ACME Inc." xml:lang="en-US"/>
371   </rim:Name>
372   <rim:Description>
373     <rim:LocalizedString value="ACME makes Java. Provider of free Java
374 software." xml:lang="en-US"/>
375   </rim:Description>
376   <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
377     identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
378     value="ACME"/>
379   </rim:ExternalIdentifier>
380
381   <!--Classify Organization as a Software Publisher using NAICS Taxonomy-->
382   <rim:Classification id=${CLASSIFICATION_ID}
383     classificationNode=${NAICS_SOFTWARE_PUBLISHER_NODE_ID}
384     classifiedObject=${ACME_ORG_ID}>
385
386 </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

398 [ebRIM] allows this scheme to be extensible.

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

402

```
403 <rim:Association  
404     id=${ASSOCIATION_ID}  
405     associationType=${CANONICAL_ASSOCIATION_TYPE_OFFERS_SERVICE_ID}  
406     sourceObject=${ACME_ORG_ID}  
407     targetObject=${ACME_SERVICE1_ID}/>
```

408

Listing 8: Example of Object Association

409 3.1.7 Object References To Web Content

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

415

```
416 <rim:ExternalLink externalURI="http://www.acme.com"  
417     id=${ACME_WEBSITE_EXTERNAL_ID}>  
418 <rim:Association  
419     id=${EXTERNALLYLINKS_ASSOCIATION_ID}  
420     associationType=${CANONICAL_ASSOCIATION_TYPE_EXTERNALLY_LINKS_ID}  
421     sourceObject=${ACME_WEBSITE_EXTERNAL_ID}  
422     targetObject=${ACME_ORG_ID}/>
```

423

Listing 9: Example of Reference to Web Content Using ExternalLink

424 3.1.8 Object Packaging

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

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

432

```
433 <rim:RegistryPackage  
434     id=${ACME_SERVICES_PACKAGE_ID}>  
435     <rim:RegistryObjectList>  
436         <rim:ObjectRef id=${ACME_SERVICE1_ID}>  
437             <rim:RegistryPackage  
438                 id=${ACME_PURCHASING_SERVICES_PACKAGE_ID}>  
439                 <rim:ObjectRef id=${ACME_PURCHASING_SERVICE1_ID}>  
440                 <rim:ObjectRef id=${ACME_PURCHASING_SERVICE2_ID}>  
441             </rim:RegistryPackage>  
442             <rim:RegistryPackage  
443                 id=${ACME_HR_SERVICES_PACKAGE_ID}>  
444                 <rim:ObjectRef id=${ACME_HR_SERVICE1_ID}>  
445                 <rim:ObjectRef id=${ACME_HR_SERVICE2_ID}>  
446             </rim:RegistryPackage>  
447         </rim:RegistryObjectList>  
448     </rim:RegistryPackage>
```

449

Listing 10: Example of Object Packaging Using RegistryPackages

450 3.1.9 ExtrinsicObject

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

455 3.1.10 Service Description

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

459 3.2 Overview of [ebRS]

460 The [ebRS] specification defines the interfaces supported by an ebXML Registry and their bindings to
461 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 procedures MUST already be available in the Registry as described in this chapter. As this profile aims to enhance ebXML registry semantics without causing any changes in the ebXML Registry architecture specification, the stored procedures proposed in this specification SHOULD be submitted to the ebXML Registry by using the Stored Query API of [ebRS]. It should be noted that arbitrary submission of the stored procedures defined in this profile would result in duplicate entries. Therefore, it would be more efficient to classify these stored procedures under a ClassificationScheme (e.g. urn:oasis:names:tc:ebxml-regrep:SemanticStoredProcedures).

The following ebRIM standard relational schema is used in coding the stored procedures 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'>
</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

512 rdfs:range, is the targetObject of the Association Type. Consider the following example which defines an
513 rdf:Property instance called "hasAirport" whose domain is "City" and whose range is "Airport" classes:

514

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

519

Example rdf:Property

520

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

526

Example: ebRIM construct Association corresponding to rdf:Property

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

529 4.1.3 rdfs:subPropertyOf → rim:Association Type subPropertyOf

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

533

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

537

Example rdfs:subPropertyOf

538 A new ebXML RIM Association Type called "Property" MUST be defined to represent rdfs:subPropertyOf
539 in ebRIM. Such a semantic enhancement brings the following processing need: given a property, it should
540 be possible to retrieve all of its super properties. This requires a recursion mechanism in SQL queries.

541 The freebXML implementation allows various relational database products such as Oracle, PostgreSQL
542 and MS SQL Server 2005 to be used as the database. These products have different support for
543 recursion mechanism in SQL Queries. The following stored procedure is for retrieving all super properties
544 of a given property instance (Association instance in ebXML terminology) recursively in a property
545 hierarchy (hierarchy of Association Types) in freebXML Registry implementations using MS SQL Server
546 2005 as the database:

547

```
548 CREATE PROCEDURE findAllSuperProperties  
549   @propertyName varchar(50)  
550 AS  
551 WITH  
552   Parents(superPropertyID) AS  
553   (  
554     SELECT A3.id  
555     FROM Association A1, Association A2, Association A3, Name_ N  
556     WHERE A2.associationType LIKE 'urn:oasis:names:tc:ebxml-  
557   regrep:AssociationType:SubPropertyOf' AND  
558     A1.id = N.parent AND N.value LIKE @propertyName AND  
559     A2.sourceObject = A1.id AND A2.targetObject = A3.id  
560   UNION ALL  
561     SELECT A.targetObject  
562     FROM Association A JOIN Parents P  
563     ON P.superPropertyID = A.sourceObject  
564     WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-  
565   regrep:AssociationType:SubPropertyOf'  
566   )
```

```
567 SELECT * FROM Parents
568 GO
```

569 **Recursive stored procedure for MS SQL Server 2005 retrieving all super classes of a given** 570 **property (Association)**

571 The following is an example on how the stored procedure `findAllSuperProperties` can be called:

```
572 <AdhocQueryRequest>
573   <query:ResponseOption returnComposedObjects="true"
574   returnType="LeafClassWithRepositoryItem"/>
575   <rim:Slot name="urn:oasis:names:tc:ebxml-
576   regrep:3.0:rs:AdhocQueryRequest:queryId">
577     <rim:ValueList>
578       <rim:Value>UUID OF THE STORED QUERY</rim:Value>
579     </rim:ValueList>
580   </rim:Slot>
581   <rim:Slot name="$propertyName ">
582     <rim:ValueList>
583       <rim:Value>%creditCardPayment%</rim:Value>
584     </rim:ValueList>
585   </rim:Slot>
586 </AdhocQueryRequest>
```

587 **Example: Executing stored procedure** `findAllSuperProperties`

588 **4.1.4 rdfs:subClassOf → rim:Association Type subClassOf**

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

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

```
599
600 <owl:Class rdf:ID="AirReservationServices">
601   <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-
602   s/1.0/Profile.owl#Profile"/>
603   <rdfs:subClassOf rdf:resource="#AirServices"/>
604 </owl:Class>
```

605 **Example rdfs:subClassOf**

606 To express this semantics through ebXML RIM constructs, "AirReservationServices" ClassificationNode is
607 associated both with the "OWL-S Profile" and "AirServices" ClassificationNodes through the "targetObject"
608 and "sourceObject" attributes of the two instances of the newly created "subClassOf" ebXML Association
609 Type as shown in the following:

```
610
611 <rim:Association id='subClassOf1' associationType='urn:oasis:names:tc:ebxml-
612   regrep:AssociationType:subClassOf'
613   sourceObject='AirReservationServices' targetObject='OWL-S Profile' >
614 </rim:Association>
615 <rim:Association id='subClassOf2' associationType='urn:oasis:names:tc:ebxml-
616   regrep:AssociationType:subClassOf'
617   sourceObject='AirReservationServices' targetObject='AirServices' >
618 </rim:Association>
```

619 Once such a semantics is defined, there is a need to process the objects in the registry according to the
620 semantics implied; that is, given a class, it should be possible to retrieve all of its subclasses and/or all of
621 its super classes. By making the required stored procedures available in the registry, this need can be
622 readily served. The following parameterized (generic) procedure MUST be available in the registry to find
623 all the immediate super classes of a given class:

624

```
625 CREATE PROCEDURE findImmediateSuperClasses($className) AS
626 BEGIN
627 SELECT C2.id
628 FROM Association A, Name N, ClassificationNode C1, ClassificationNode C2
629 WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
630 regrep:AssociationType:subClassOf' AND
631 C1.id = N.parent AND
632 N.value LIKE $className AND
633 A.sourceObject = C1.id AND
634 A.targetObject = C2.id
635 END;
```

636 **Parameterized (generic) stored procedure retrieving immediate super classes of a given**
637 **classification node**

638

639 The following is an example on how this stored procedure can be called:

```
640 <AdhocQueryRequest>
641 <query:ResponseOption returnComposedObjects="true"
642 returnType="LeafClassWithRepositoryItem"/>
643 <rim:Slot name="urn:oasis:names:tc:ebxml-
644 regrep:3.0:rs:AdhocQueryRequest:queryId">
645 <rim:ValueList>
646 <rim:Value>UUID OF THE STORED QUERY</rim:Value>
647 </rim:ValueList>
648 </rim:Slot>
649 <rim:Slot name="$className ">
650 <rim:ValueList>
651 <rim:Value>%AirReservationServices%</rim:Value>
652 </rim:ValueList>
653 </rim:Slot>
654 </AdhocQueryRequest>
```

655 The following procedure MUST be available in the registry to find all the immediate subclasses of a given
656 class:

657

```
658 CREATE PROCEDURE findImmediateSubClasses($className) AS
659 BEGIN
660 SELECT C2.id
661 FROM Association A, Name N, ClassificationNode C1, ClassificationNode C2
662 WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
663 regrep:AssociationType:subClassOf' AND
664 C1.id = N.parent AND
665 N.value LIKE $className AND
666 A.sourceObject = C2.id AND
667 A.targetObject = C1.id
668 END;
```

669 **Parameterized (generic) stored procedure retrieving immediate subclasses of a given**
670 **classification node**

671 The following is an example on how this stored procedure can be called:

```
672 <AdhocQueryRequest>
673 <query:ResponseOption returnComposedObjects="true"
674 returnType="LeafClassWithRepositoryItem"/>
675 <rim:Slot name="urn:oasis:names:tc:ebxml-
676 regrep:3.0:rs:AdhocQueryRequest:queryId">
677 <rim:ValueList>
678 <rim:Value> UUID OF THE STORED QUERY </rim:Value>
679 </rim:ValueList>
680 </rim:Slot>
681 <rim:Slot name="$className ">
682 <rim:ValueList>
683 <rim:Value>%AirServices%</rim:Value>
684 </rim:ValueList>
```

```
685     </rim:Slot>
686 </AdhocQueryRequest>
```

687 It should be noted that, given a class, finding its immediate subclasses, super classes is necessary but not
688 sufficient. Given a class, it should be possible to retrieve all of its subclasses, and all of its super classes.
689 This requires a recursion mechanism in SQL queries. The freebXML implementation allows various
690 relational database products such as Oracle, PostgreSQL and MS SQL Server 2005 to be used as the
691 database. These products have different support for recursion mechanisms in SQL Queries. The following
692 stored procedure is for retrieving recursively all super classes of a given ClassificationNode in freebXML
693 Registry implementations using MS SQL Server 2005 as the database:

```
694
695 CREATE PROCEDURE findAllSuperClasses
696     @className varchar(50)
697 AS
698 WITH
699     Parents(superClassID) AS
700     (
701         SELECT C2.id
702         FROM Association A, Name_ N, ClassificationNode C1,
703         ClassificationNode C2
704         WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
705         regrep:AssociationType:SubClassOf' AND
706         C1.id = N.parent AND N.value LIKE @className AND
707         A.sourceObject = C1.id AND A.targetObject = C2.id
708     UNION ALL
709     SELECT A.targetObject
710     FROM Association A JOIN Parents P
711     ON P.superClassID = A.sourceObject
712     WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
713     regrep:AssociationType:SubClassOf'
714     )
715 SELECT * FROM Parents
716 GO
```

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

719 The following is an example on how the stored procedure `findAllSuperClasses` can be called:

```
720 <AdhocQueryRequest>
721   <query:ResponseOption returnComposedObjects="true"
722   returnType="LeafClassWithRepositoryItem"/>
723   <rim:Slot name="urn:oasis:names:tc:ebxml-
724   regrep:3.0:rs:AdhocQueryRequest:queryId">
725     <rim:ValueList>
726       <rim:Value>UUID OF THE STORED QUERY</rim:Value>
727     </rim:ValueList>
728   </rim:Slot>
729   <rim:Slot name="$className ">
730     <rim:ValueList>
731       <rim:Value>%AirReservationServices%</rim:Value>
732     </rim:ValueList>
733   </rim:Slot>
734 </AdhocQueryRequest>
```

735 **Example: Executing stored procedure `findAllSuperClasses`**

736 The following stored procedure is for retrieving all subclasses recursively of a given ClassificationNode in
737 freebXML Registry implementations using MS SQL Server 2005 as the database:

```
738
739 CREATE PROCEDURE findAllSubClasses
740     @className varchar(50)
741 AS
742 WITH
743     Children(subClassID) AS
744     (
```

```

745         SELECT C1.id
746         FROM Association A, Name_ N, ClassificationNode C1,
747 ClassificationNode C2
748         WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
749 regrep:AssociationType:SubClassOf' AND
750         C2.id = N.parent AND N.value LIKE @className AND
751         A.sourceObject = C1.id AND A.targetObject = C2.id
752     UNION ALL
753     SELECT A.sourceObject
754     FROM Association A JOIN Children C
755     ON C.subClassID = A.targetObject
756     WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
757 regrep:AssociationType:SubClassOf'
758 )
759 SELECT * FROM Children
760 GO

```

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

763 The following is an example on how the stored procedure `findAllSubClasses` can be called:

```

764
765 <AdhocQueryRequest>
766   <query:ResponseOption returnComposedObjects="true"
767   returnType="LeafClassWithRepositoryItem"/>
768   <rim:Slot name="urn:oasis:names:tc:ebxml-
769   regrep:3.0:rs:AdhocQueryRequest:queryId">
770     <rim:ValueList>
771       <rim:Value>UUID OF THE STORED QUERY</rim:Value>
772     </rim:ValueList>
773   </rim:Slot>
774   <rim:Slot name="$className ">
775     <rim:ValueList>
776       <rim:Value>%AirServices%</rim:Value>
777     </rim:ValueList>
778   </rim:Slot>
779 </AdhocQueryRequest>

```

780 **Example: Executing stored procedure** `findAllSubClasses`

781 4.1.5 owl:Individual → rim:ExtrinsicObject

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

```

785
786 <owl:Class rdf:ID="TravelWebService">
787 </owl:Class>
788
789 <owl:ObjectProperty rdf:ID="paymentMethod">
790   <rdfs:domain rdf:resource="#TravelWebService"/>
791   <rdfs:range rdf:resource="#PossiblePaymentMethods"/>
792 </owl:ObjectProperty >

```

793 **Example owl:Class example**

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

```

798
799 <TravelWebService rdf:ID="MyTravelWebService">
800 <paymentMethod> Cash </paymentMethod>
801 </TravelWebService>

```

802 **Example owl:Individual example**

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

807 **4.2 Representing OWL (In)Equality Constructs in ebXML RIM**

808 **4.2.1 owl:equivalentClass, owl:equivalentProperty → rim:Association Type**
809 **EquivalentTo**

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

814 The following stored procedure MUST be available in the registry to retrieve all the equivalent classes of a
815 given ClassificationNode:

```
816 CREATE PROCEDURE findEquivalentClasses($className) BEGIN  
817     SELECT A.targetObject  
818     FROM Association A, Name N, ClassificationNode C  
819     WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-  
820 regrep:AssociationType:EquivalentTo' AND  
821     C.id = N.parent AND  
822     N.value LIKE $className AND  
823     A.sourceObject = C.id  
824  
825 END;
```

826 **Parameterized (generic) stored procedure retrieving all the equivalent classes of a given**
827 **classification node**

828 The following is an example on how this stored procedure can be called:

```
829 <AdhocQueryRequest>  
830   <query:ResponseOption returnComposedObjects="true"  
831   returnType="LeafClassWithRepositoryItem"/>  
832   <rim:Slot name="urn:oasis:names:tc:ebxml-  
833   regrep:3.0:rs:AdhocQueryRequest:queryId">  
834     <rim:ValueList>  
835       <rim:Value> UUID OF THE STORED QUERY </rim:Value>  
836     </rim:ValueList>  
837   </rim:Slot>  
838   <rim:Slot name="$className ">  
839     <rim:ValueList>  
840       <rim:Value>%AirServices%</rim:Value>  
841     </rim:ValueList>  
842   </rim:Slot>  
843 </AdhocQueryRequest>
```

844 The following stored procedure MUST be available in the registry to retrieve all the equivalent properties
845 (Association Type) of a given property (Association Type):

```
846 CREATE PROCEDURE findEquivalentProperties($propertyName) BEGIN  
847     SELECT A1.targetObject  
848     FROM Association A1, Name N, Association A2  
849     WHERE A1.associationType LIKE 'urn:oasis:names:tc:ebxml-  
850 regrep:AssociationType:EquivalentTo' AND  
851     A2.id = N.parent AND  
852     N.value LIKE $propertyName AND  
853     A1.sourceObject = A2.id  
854  
855 END;
```

856 **Parameterized (generic) stored procedure retrieving all the equivalent Association Type of a**

857 **given Association Type**

858 4.2.2 owl:sameAs → rim:Association Type sameAs

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

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

863

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

867 **Example owl:sameAs**

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

870 Furthermore, the following stored procedure MUST be available in the registry to retrieve all the
871 "ExtrinsicObjects" defined to be the same with a given ExtrinsicObject:

872

```
873 CREATE PROCEDURE findTheSameExtrinsicObjects($extrinsicObjectName) BEGIN  
874   SELECT A.targetObject  
875   FROM Association A, Name N, ExtrinsicObject E  
876   WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-  
877   regrep:AssociationType:sameAs' AND  
878   E.id = N.parent AND  
879   N.value LIKE $ extrinsicObjectName AND  
880   A.sourceObject = E.id  
881  
882 END;
```

883 **Parameterized (generic) stored procedure retrieving all the "ExtrinsicObjects" defined to be** 884 **the same with a given ExtrinsicObject**

885 The following is an example on how this stored procedure can be called:

```
886 <AdhocQueryRequest>  
887   <query:ResponseOption returnComposedObjects="true"  
888   returnType="LeafClassWithRepositoryItem"/>  
889   <rim:Slot name="urn:oasis:names:tc:ebxml-  
890   regrep:3.0:rs:AdhocQueryRequest:queryId">  
891     <rim:ValueList>  
892       <rim:Value> UUID OF THE STORED QUERY </rim:Value>  
893     </rim:ValueList>  
894   </rim:Slot>  
895   <rim:Slot name="$extrinsicObjectName ">  
896     <rim:ValueList>  
897       <rim:Value>%MyDocument%</rim:Value>  
898     </rim:ValueList>  
899   </rim:Slot>  
900 </AdhocQueryRequest>
```

901 4.2.3 owl:differentFrom → rim:Association Type differentFrom

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

906

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


910 Example owl:differentFrom

911 This translates into declaring two "ExtrinsicObjects" in the ebXML registry to be different from each other.
912 For this purpose a new Association Type "differentFrom" MUST be defined in the ebXML registry to
913 explicitly indicate that the sourceRegistryObject is different from the targetRegistryObject.

914

```
915 CREATE PROCEDURE findDifferentExtrinsicObjects($extrinsicObjectName)
916 BEGIN
917     SELECT A.targetObject
918     FROM Association A, Name N, ExtrinsicObject E
919     WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
920 regrep:AssociationType:differentFrom' AND
921     E.id = N.parent AND
922     N.value LIKE $ extrinsicObjectName AND
923     A.sourceObject = E.id
924
925 END;
```

926 Parameterized (generic) stored procedure retrieving all the "ExtrinsicObjects" defined to be 927 different from a given ExtrinsicObject

928 The following is an example on how this stored procedure can be called:

```
929 <AdhocQueryRequest>
930   <query:ResponseOption returnComposedObjects="true"
931   returnType="LeafClassWithRepositoryItem"/>
932   <rim:Slot name="urn:oasis:names:tc:ebxml-
933   regrep:3.0:rs:AdhocQueryRequest:queryId">
934     <rim:ValueList>
935       <rim:Value> UUID OF THE STORED QUERY </rim:Value>
936     </rim:ValueList>
937   </rim:Slot>
938   <rim:Slot name="$extrinsicObjectName ">
939     <rim:ValueList>
940       <rim:Value>%MyDocument%</rim:Value>
941     </rim:ValueList>
942   </rim:Slot>
943 </AdhocQueryRequest>
```

944 4.2.4 owl:AllDifferent

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

949 The following example states that the three instances of the "WebService" collection are all different from
950 one another:

```
951 <owl:AllDifferent>
952   <owl:distinctMembers rdf:parseType="Collection">
953     <WebService rdf:about="#MyCarService"/>
954     <WebService rdf:about="#MyFlightService"/>
955     <WebService rdf:about="#MyHotelService"/>
956   </owl:distinctMembers>
957 </owl:AllDifferent>
```

958 Example owl:AllDifferentFrom

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

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

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

968

```
969 <rim:RegistryPackage id = "CollectionRegistryPackage" >  
970   <rim:Name>  
971     <rim:LocalizedString value = "Collection"/>  
972   </rim:Name>  
973   <rim:Slot name="allDifferent">  
974     <rim:ValueList>  
975       <rim:Value>>true</rim:Value>  
976     </rim:ValueList>  
977   </rim:Slot>  
978 </rim:RegistryPackage>  
979  
980 <rim:Association id = "CollectionRegistryPackageAssoc1"  
981 associationType =  
982 "urn:oasis:names:tc:ebxmlregrep:AssociationType:HasMember"  
983 sourceObject = "CollectionRegistryPackage"  
984 targetObject = "MyCarService" />  
985  
986 <rim:Association id = "CollectionRegistryPackageAssoc2"  
987 associationType =  
988 "urn:oasis:names:tc:ebxmlregrep:AssociationType:HasMember"  
989 sourceObject = "CollectionRegistryPackage"  
990 targetObject = "MyFlightService" />  
991  
992 <rim:Association id = "CollectionRegistryPackageAssoc3"  
993 associationType =  
994 "urn:oasis:names:tc:ebxmlregrep:AssociationType:HasMember"  
995 sourceObject = "CollectionRegistryPackage"  
996 targetObject = "MyHotelService" />
```

997

```
998 CREATE PROCEDURE findAllDifferent($registryObjectName)  
999 BEGIN  
1000 SELECT A2.targetObject  
1001 FROM Association A1, Association A2, Name_ N, RegistryObject RO,  
1002 RegistryPackage RP, Slot S  
1003 WHERE A1.associationType LIKE 'urn:oasis:names:tc:ebxml-regrep:  
1004 AssociationType:HasMember' AND  
1005 RO.id = N.parent AND  
1006 N.value LIKE $registryObjectName AND  
1007 A1.sourceObject = RP.id AND  
1008 S.parent = RP.id AND  
1009 S.name_ LIKE 'allDifferent' AND S.value LIKE 'true' AND  
1010 A1.targetObject = RO.id AND  
1011 A2.associationType LIKE 'urn:oasis:names:tc:ebxml-regrep:  
1012 AssociationType:HasMember' AND  
1013 A2.sourceObject = RP.id AND  
1014 A2.targetObject != RO.id  
1015 END;
```

1016

1017 4.3 Representing OWL Property Characteristics in ebRIM

1018 4.3.1 owl:ObjectProperty → rim:Association Type objectProperty

1019 To represent OWL ObjectProperty in ebXML, a new type of Association called "ObjectProperty" MUST be
1020 defined. Consider the following example which defines an object property "hasAirport" whose domain is
1021 "City" and whose range is "Airport":

1022

```
1023 <owl:ObjectProperty rdf:ID="hasAirport">  
1024   <rdfs:domain rdf:resource="#City"/>
```

```
1025 <rdfs:range rdf:resource="#AirPort"/>
1026 </owl:ObjectProperty>
```

Example owl:ObjectProperty

```
1028
1029 <rim:Association id='hasAirport' associationType='urn:oasis:names:tc:ebxml-
1030 regrep:AssociationType:objectProperty'
1031 sourceObject= 'City' targetObject='Airport' >
1032 </rim:Association>
```

Example Corresponding ebRIM construct Association

1034 Once such objectProperty definitions are stored in the ebXML registry, they can be retrieved through
1035 ebXML query facilities by the user. However, the following parameterized (generic) stored procedure
1036 MUST be available in the registry to facilitate this access.

```
1037
1038 CREATE PROCEDURE findObjectProperties($className) AS
1039 BEGIN
1040 SELECT A.id
1041 FROM Association A, Name_ N, ClassificationNode C
1042 WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1043 regrep:AssociationType:objectProperty' AND
1044 C.id = N.parent AND
1045 N.value LIKE $className AND
1046 A.sourceObject = C.id
1047 END;
```

Parameterized (generic) stored procedure retrieving all the object properties of a given classification node

1048 The following is an example on how this stored procedure can be called:

```
1049
1050 <AdhocQueryRequest>
1051 <query:ResponseOption returnComposedObjects="true"
1052 returnType="LeafClassWithRepositoryItem"/>
1053 <rim:Slot name="urn:oasis:names:tc:ebxml-
1054 regrep:3.0:rs:AdhocQueryRequest:queryId">
1055 <rim:ValueList>
1056 <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1057 </rim:ValueList>
1058 </rim:Slot>
1059 <rim:Slot name="$className ">
1060 <rim:ValueList>
1061 <rim:Value>%AirServices%</rim:Value>
1062 </rim:ValueList>
1063 </rim:Slot>
1064 </AdhocQueryRequest>
```

1066 The following procedure MUST be available in the registry to be used to retrieve all of the properties of a
1067 given class including the ones inherited from its immediate super classes:

```
1068
1069 CREATE PROCEDURE findImmediateInheritedObjectProperties ($className) AS
1070 SELECT A.id FROM Association A, ClassificationNode C WHERE
1071 A.sourceObject=C.id AND
1072 A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1073 regrep:AssociationType:objectProperty' AND
1074 C.id IN (
1075 SELECT parent
1076 FROM name_
1077 WHERE value LIKE $className
1078 UNION
1079 findSuperClasses($className)
1080 )
1081 END;
```

Parameterized (generic) stored procedure retrieving all of the properties of a given classification node including the ones inherited from its immediate super classes

1084 The following is an example on how this stored procedure can be called:

```

1085 <AdhocQueryRequest>
1086   <query:ResponseOption returnComposedObjects="true"
1087   returnType="LeafClassWithRepositoryItem"/>
1088   <rim:Slot name="urn:oasis:names:tc:ebxml-
1089   regrep:3.0:rs:AdhocQueryRequest:queryId">
1090     <rim:ValueList>
1091       <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1092     </rim:ValueList>
1093   </rim:Slot>
1094   <rim:Slot name="$className ">
1095     <rim:ValueList>
1096       <rim:Value>%AirReservationServices%</rim:Value>
1097     </rim:ValueList>
1098   </rim:Slot>
1099 </AdhocQueryRequest>

```

1100 It should be noted that, given a class, finding the object properties inherited from immediate super classes
1101 is necessary but not sufficient. Given a class, it should be possible to retrieve all of the object properties
1102 inherited from its super classes. This requires a recursion mechanism in SQL queries. The freebXML
1103 implementation allows various relational database products such as Oracle, PostgreSQL and MS SQL
1104 Server 2005 to be used as the database. These products have different support for recursion in SQL
1105 Queries. The following stored procedure is for retrieving all inherited ObjectProperties recursively of a
1106 given ClassificationNode in a ClassificationScheme in freebXML Registry implementations using MS SQL
1107 Server 2005 as the database:

```

1108
1109 CREATE PROCEDURE findAllInheritedObjectProperties
1110   @className varchar(50)
1111 AS
1112 WITH Parents(superClassID) AS (
1113   SELECT C2.id
1114   FROM Association A, Name_ N, ClassificationNode C1,
1115   ClassificationNode C2
1116   WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1117   regrep:AssociationType:SubClassOf' AND
1118   C1.id = N.parent AND N.value LIKE @className AND
1119   A.sourceObject = C1.id AND A.targetObject = C2.id
1120 UNION ALL
1121   SELECT A.targetObject
1122   FROM Association A JOIN Parents P
1123   ON P.superClassID = A.sourceObject
1124   WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1125   regrep:AssociationType:SubClassOf'
1126 ) SELECT A.id
1127   FROM Association A, Parents P
1128   WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1129   regrep:AssociationType:ObjectProperty' AND
1130   A.sourceObject=P.superClassID
1131 UNION
1132
1133 SELECT A.id
1134 FROM Name_ N, ClassificationNode C, Association A
1135 WHERE C.id = N.parent AND N.value LIKE @className AND
1136   A.sourceObject=C.id AND A.associationType LIKE
1137   'urn:oasis:names:tc:ebxml-regrep:AssociationType:ObjectProperty'
1138 GO

```

1139 **Recursive stored procedure for MS SQL Server 2005 database retrieving all inherited Object**
1140 **Properties of a given classification node**

1141 The following is an example on how the stored procedure `findAllInheritedObjectProperties` can be
1142 called:

```

1143 <AdhocQueryRequest>
1144   <query:ResponseOption returnComposedObjects="true"
1145   returnType="LeafClassWithRepositoryItem"/>

```

```

1146     <rim:Slot name="urn:oasis:names:tc:ebxml-
1147     regrep:3.0:rs:AdhocQueryRequest:queryId">
1148       <rim:ValueList>
1149         <rim:Value>UUID OF THE STORED QUERY</rim:Value>
1150       </rim:ValueList>
1151     </rim:Slot>
1152     <rim:Slot name="$className ">
1153       <rim:ValueList>
1154         <rim:Value>%AirReservationServices%</rim:Value>
1155       </rim:ValueList>
1156     </rim:Slot>
1157   </AdhocQueryRequest>

```

1158 **Example: Executing stored procedure** findAllInheritedObjectProperties

1159 4.3.2 owl:DatatypeProperty → rim:Association Type DatatypeProperty

1160 Similarly, to represent OWL DatatypeProperty in ebXML, a new Association Type called
 1161 "DatatypeProperty" MUST be defined. Consider the following example which defines an datatype property
 1162 "hasPrice" whose domain is the "AirReservationServices" and whose range is "XMLSchema
 1163 nonNegativeInteger". How OWL XML Schema types are handled in ebXML RIM is described in Section
 1164 4.9.

```

1165 <owl:DatatypeProperty rdf:ID="hasPrice">
1166   <rdfs:subpropertyOf rdf:resource="http://www.daml.org/services/daml-
1167   s/2001/05/Profile.owl"/>
1168   <rdfs:domain rdf:resource="#AirReservationServices"/>
1169   <rdfs:range
1170   rdf:resource="http://www.w3.org/2000/10/XMLSchema/nonNegativeInteger"/>
1171 </owl:DatatypeProperty>

```

1172 **Example owl:DatatypeProperty**

1173 The following parameterized (generic) stored procedure MUST be available in the registry to facilitate the
 1174 direct access to datatype properties of a given classification node.

```

1175 CREATE PROCEDURE findDatatypeProperties($className) AS
1176 BEGIN
1177 SELECT A.id
1178 FROM Association A, Name_ N, ClassificationNode C
1179 WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1180 regrep:AssociationType:datatypeProperty' AND
1181        C.id = N.parent AND
1182        N.value LIKE $className AND
1183        A.sourceObject = C.id
1184 END;
1185

```

1186 **Parameterized (generic) stored procedure retrieving all the datatype properties of a given**
 1187 **classification node**

1188 The following is an example on how this stored procedure can be called:

```

1189 <AdhocQueryRequest>
1190   <query:ResponseOption returnComposedObjects="true"
1191   returnType="LeafClassWithRepositoryItem"/>
1192   <rim:Slot name="urn:oasis:names:tc:ebxml-
1193   regrep:3.0:rs:AdhocQueryRequest:queryId">
1194     <rim:ValueList>
1195       <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1196     </rim:ValueList>
1197   </rim:Slot>
1198   <rim:Slot name="$className ">
1199     <rim:ValueList>
1200       <rim:Value>%AirReservationServices%</rim:Value>
1201     </rim:ValueList>
1202   </rim:Slot>
1203 </AdhocQueryRequest>

```

1204 It should be noted that, given a class, finding the datatype properties inherited from immediate super
 1205 classes is necessary but not sufficient. Given a class, it should be possible to retrieve all of the datatype

1206 properties inherited from its super classes. This requires a recursion mechanism in SQL queries. The
 1207 freebXML implementation allows various relational database products such as Oracle, PostgreSQL and
 1208 MS SQL Server 2005 to be used as the database. These products have different support for recursion in
 1209 SQL Queries. The following stored procedure is for retrieving all inherited DatatypeProperties recursively
 1210 of a given ClassificationNode in a ClassificationScheme in freebXML Registry implementations using MS
 1211 SQL Server 2005 as the database:

```

1212
1213 CREATE PROCEDURE findAllInheritedDatatypeProperties
1214     @className varchar(50)
1215 AS
1216 WITH Parents(superClassID) AS (
1217     SELECT C2.id
1218     FROM Association A, Name_ N, ClassificationNode C1,
1219     ClassificationNode C2
1220     WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1221     regrep:AssociationType:SubClassOf' AND
1222     C1.id = N.parent AND N.value LIKE @className AND
1223     A.sourceObject = C1.id AND A.targetObject = C2.id
1224 UNION ALL
1225     SELECT A.targetObject
1226     FROM Association A JOIN Parents P
1227     ON P.superClassID = A.sourceObject
1228     WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1229     regrep:AssociationType:SubClassOf'
1230 ) SELECT A.id
1231     FROM Association A, Parents P
1232     WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1233     regrep:AssociationType:DatatypeProperty' AND
1234     A.sourceObject=P.superClassID
1235 UNION
1236
1237 SELECT A.id
1238 FROM Name_ N, ClassificationNode C, Association A
1239 WHERE C.id = N.parent AND N.value LIKE @className AND
1240     A.sourceObject=C.id AND A.associationType LIKE
1241     'urn:oasis:names:tc:ebxml-regrep:AssociationType:DatatypeProperty'
1242 GO
  
```

1243 **Recursive stored procedure for MS SQL Server 2005 database retrieving all inherited Datatype**
 1244 **Properties of a given classification node**

1245 The following is an example on how the stored procedure findAllInheritedDatatypeProperties can be
 1246 called:

```

1247 <AdhocQueryRequest>
1248   <query:ResponseOption returnComposedObjects="true"
1249   returnType="LeafClassWithRepositoryItem"/>
1250   <rim:Slot name="urn:oasis:names:tc:ebxml-
1251   regrep:3.0:rs:AdhocQueryRequest:queryId">
1252     <rim:ValueList>
1253       <rim:Value>UUID OF THE STORED QUERY</rim:Value>
1254     </rim:ValueList>
1255   </rim:Slot>
1256   <rim:Slot name="$className ">
1257     <rim:ValueList>
1258       <rim:Value>%AirReservationServices%</rim:Value>
1259     </rim:ValueList>
1260   </rim:Slot>
1261 </AdhocQueryRequest>
  
```

1262 **Example: Executing stored procedure findAllInheritedDatatypeProperties**

1263 **4.3.3 owl:TransitiveProperty → rim:Association Type transitiveProperty**

1264 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)
 1265 [McGuinness, Harmelen]. Transitive property is a subproperty of ObjectProperty and MUST be defined as
 1266 a new Association Type called "transitiveProperty" in eBRIM.

1267 Consider the following example where "succeeds" is defined as a transitive property of
1268 "TravelWebService" class:
1269

```
1270 <owl:ObjectProperty rdf:ID="succeeds">  
1271   <rdf:type rdf:resource="&owl;TransitiveProperty" />  
1272   <rdfs:domain rdf:resource="#TravelWebService" />  
1273   <rdfs:range rdf:resource="#TravelWebService" />  
1274 </owl:ObjectProperty>
```

1275 **Example owl:TransitiveProperty**

1276 Assume the following two definitions which declare three Web service instances from TravelWebService
1277 class where "MyHotelAvailabilityService" service succeeds "MyAirReservationService" and
1278 "MyInsuranceService" succeeds MyHotelAvailabilityService". Since "succeeds" is a transitive property, it
1279 follows that "MyInsuranceService" succeeds "MyAirReservationService" although this fact is not explicitly
1280 stated.

```
1281  
1282 <TravelWebService rdf:ID="MyHotelAvailabilityService">  
1283   <succeeds rdf:resource="#MyAirReservationService" />  
1284 </TravelWebService>  
1285  
1286 <TravelWebService rdf:ID="MyInsuranceService">  
1287   <succeeds rdf:resource="#MyHotelAvailabilityService" />  
1288 </TravelWebService>
```

1289 **Example owl:TransitiveProperty instances**

1290 To make any use of this transitive property in ebXML registries, coding is necessary to find out the implied
1291 information. The following stored procedure MUST be available in the registry to handle this semantics:
1292 Given a class which is a source of a transitive property, this stored procedure retrieves not only the target
1293 of a given transitive property, but if the target objects have the same property, it also retrieves their target
1294 objects too.

```
1295  
1296 CREATE PROCEDURE findTransitiveRelationships($className,$propertyName)  
1297 BEGIN SELECT A2.targetObject FROM Association A1,  
1298 Association A2, Name N1,Name N2, Name N3 WHERE  
1299 A1.associationType LIKE 'urn:oasis:names:tc:ebxml-  
1300 regrep:AssociationType:transitiveProperty' AND  
1301 A1.id = N1.parent AND  
1302 N1.value LIKE $propertyName AND  
1303 A1.sourceObject = N3.parent AND  
1304 N3.value LIKE $className AND  
1305 A2.sourceObject = A1.targetObject AND  
1306 A2.id = N2.parent AND  
1307 N2.value LIKE $propertyName AND  
1308 A2.associationType LIKE 'urn:oasis:names:tc:ebxml-  
1309 regrep:AssociationType:transitiveProperty'  
1310 UNION  
1311 SELECT A1.targetObject  
1312 FROM Association A1, Name N1, Name N3  
1313 WHERE A1.associationType LIKE 'urn:oasis:names:tc:ebxml-  
1314 regrep:AssociationType:transitiveProperty' AND  
1315 A1.id = N1.parent AND  
1316 N1.value LIKE $propertyName AND  
1317 A1.sourceObject = N3.parent AND  
1318 N3.value LIKE $className  
1319 END;
```

1320 **Parameterized (generic) stored procedure retrieving the objects in transitive relationship a** 1321 **given object**

1322 The following is an example on how this stored procedure can be called:

```
1323 <AdhocQueryRequest>
```

```

1324     <query:ResponseOption returnComposedObjects="true"
1325     returnType="LeafClassWithRepositoryItem"/>
1326     <rim:Slot name="urn:oasis:names:tc:ebxml-
1327     regrep:3.0:rs:AdhocQueryRequest:queryId">
1328         <rim:ValueList>
1329             <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1330         </rim:ValueList>
1331     </rim:Slot>
1332     <rim:Slot name="$className ">
1333         <rim:ValueList>
1334             <rim:Value>%AirReservationServices%</rim:Value>
1335         </rim:ValueList>
1336     </rim:Slot>
1337     <rim:Slot name="$propertyName ">
1338         <rim:ValueList>
1339             <rim:Value>%succeeds%</rim:Value>
1340         </rim:ValueList>
1341     </rim:Slot>
1342 </AdhocQueryRequest>

```

1343 4.3.4 owl:inverseOf → rim:Association Type inverseOf

1344 In OWL, one property may be stated to be the inverse of another property. If the property P1 is stated to
 1345 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
 1346 P1 property [McGuinness, Harmelen].

1347 4.3.4.1 Retrieving the Target Objects of a given Association of a given 1348 ClassificationNode

1349 The following stored procedure MUST be available in the ebXML Registry to retrieve the targetObjects
 1350 from the Registry, given a sourceObject and an Association Type.

```

1351
1352 CREATE PROCEDURE findTargetObjects($className, $propertyName)
1353 BEGIN
1354 SELECT C2.id
1355 FROM Association A, Name_ N, Name_ N2, ClassificationNode C1,
1356 ClassificationNode C2
1357 WHERE A.id=N2.parent AND
1358       N2.value LIKE $propertyName AND
1359       C1.id = N.parent AND
1360       N.value LIKE $className AND
1361       A.sourceObject = C1.id AND
1362       A.targetObject = C2.id
1363 END;

```

1364 **Parameterized (generic) Stored Procedure retrieving the Target Objects from the Registry, 1365 given a Source Object and an Association**

1366 The following is an example on how this stored procedure can be called:

```

1367 <AdhocQueryRequest>
1368     <query:ResponseOption returnComposedObjects="true"
1369     returnType="LeafClassWithRepositoryItem"/>
1370     <rim:Slot name="urn:oasis:names:tc:ebxml-
1371     regrep:3.0:rs:AdhocQueryRequest:queryId">
1372         <rim:ValueList>
1373             <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1374         </rim:ValueList>
1375     </rim:Slot>
1376     <rim:Slot name="$className ">
1377         <rim:ValueList>
1378             <rim:Value>%AirReservationServices%</rim:Value>
1379         </rim:ValueList>
1380     </rim:Slot>
1381     <rim:Slot name="$propertyName ">

```



```

1382     <rim:ValueList>
1383         <rim:Value>%paymentMethod%</rim:Value>
1384     </rim:ValueList>
1385     </rim:Slot>
1386 </AdhocQueryRequest>

```

1387 4.3.4.2 Retrieving the Target Objects from the Registry which are in "inverseOf" 1388 relationship to a given Association of a given ClassificationNode

1389 The following stored procedure MUST be available in the ebXML Registry to retrieve the target objects
1390 from the Registry, which are in "inverseOf" relationship to a given Association of a given
1391 ClassificationNode.

1392

```

1393 CREATE PROCEDURE findTOinverseOf($className, $propertyName)
1394 BEGIN
1395 SELECT A3.sourceObject
1396 FROM Association A1, Association A2, Association A3, Name_ N, NAME_ N2,
1397 ClassificationNode C1
1398 WHERE A2.associationType LIKE 'urn:oasis:names:tc:ebxml-
1399 regrep:AssociationType:inverseOf' AND
1400     A1.id = N.parent AND
1401     N.value LIKE $propertyName AND
1402     A2.sourceObject = A1.id AND
1403     A3.id=A2.targetObject AND
1404     C1.id = N2.parent AND
1405     N2.value LIKE $className AND
1406     A3.targetObject = C1.id
1407 END;

```

1408 Parameterized (generic) Stored Procedure retrieving the Target Objects from the Registry which 1409 are in "inverseOf" relationship to a given Association of a given ClassificationNode

1410 The following is an example on how this stored procedure can be called:

```

1411 <AdhocQueryRequest>
1412     <query:ResponseOption returnComposedObjects="true"
1413     returnType="LeafClassWithRepositoryItem"/>
1414     <rim:Slot name="urn:oasis:names:tc:ebxml-
1415     regrep:3.0:rs:AdhocQueryRequest:queryId">
1416         <rim:ValueList>
1417             <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1418         </rim:ValueList>
1419     </rim:Slot>
1420     <rim:Slot name="$className ">
1421         <rim:ValueList>
1422             <rim:Value>%AirReservationServices%</rim:Value>
1423         </rim:ValueList>
1424     </rim:Slot>
1425     <rim:Slot name="$propertyName ">
1426         <rim:ValueList>
1427             <rim:Value>%succeeds%</rim:Value>
1428         </rim:ValueList>
1429     </rim:Slot>
1430 </AdhocQueryRequest>

```

1431 4.3.4.3 A Clarifying Example

1432 Consider, for example, the "succeeds" property defined in Section 4.3.3. To denote that a certain Web
1433 service instance precedes another during execution, we may define the "precedes" property as an inverse
1434 of the "succeeds" property as follows:

1435

```

1436 <owl:ObjectProperty rdf:ID="precedes">
1437     <owl:inverseOf rdf:resource="#succeeds" />
1438 </owl:ObjectProperty>

```

1439 **Example owl:inverseOf Property**

1440 Assume that we want to find all the Web services which can succeed a given Web service. In such a
1441 case, we need not only find all the Web services which succeeds this given Web service, that is the target
1442 objects of "succeeds" Association instance, but we also need to find all the sourceObjects of the
1443 "precedes" Association instance since "precedes" is declared to be the "inverseOf" succeeds Association
1444 instance. Then the following stored procedure which is the union of the stored procedures given in Section
1445 4.9.1 and 4.9.2, gives the desired result. In other words, by using the following stored procedure, we can
1446 find all the services that precede a given service by also making use of its "succeeds" property and hence
1447 it MUST be available in the Registry.

```
1448 CREATE PROCEDURE findInverseRanges($className, $propertyName)
1449 BEGIN
1450 SELECT C2.id
1451 FROM Association A, Name_ N, Name_ N2, ClassificationNode C1,
1452 ClassificationNode C2
1453 WHERE A.id=N2.parent AND
1454 N2.value LIKE $propertyName AND
1455 C1.id = N.parent AND
1456 N.value LIKE $className AND
1457 A.sourceObject = C1.id AND
1458 A.targetObject = C2.id
1459 UNION
1460 SELECT A3.sourceObject
1461 FROM Association A1, Association A2, Association A3, Name_ N, NAME_ N2,
1462 ClassificationNode C1
1463 WHERE A2.associationType LIKE 'urn:oasis:names:tc:ebxml-
1464 regrep:AssociationType:inverseOf' AND
1465 A1.id = N.parent AND
1466 N.value LIKE $propertyName AND
1467 A2.sourceObject = A1.id AND
1468 A3.id=A2.targetObject AND
1469 C1.id = N2.parent AND
1470 N2.value LIKE $className AND
1471 A3.targetObject = C1.id
1472 END;
```

1473 **Retrieving both the Target Objects of a given Association and the Source Objects of an** 1474 **Association which is in "inverseOf" relationship to this Association**

1475 The following is an example on how this stored procedure can be called:

```
1476 <AdhocQueryRequest>
1477 <query:ResponseOption returnComposedObjects="true"
1478 returnType="LeafClassWithRepositoryItem"/>
1479 <rim:Slot name="urn:oasis:names:tc:ebxml-
1480 regrep:3.0:rs:AdhocQueryRequest:queryId">
1481 <rim:ValueList>
1482 <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1483 </rim:ValueList>
1484 </rim:Slot>
1485 <rim:Slot name="$className ">
1486 <rim:ValueList>
1487 <rim:Value>%AirReservationServices%</rim:Value>
1488 </rim:ValueList>
1489 </rim:Slot>
1490 <rim:Slot name="$propertyName ">
1491 <rim:ValueList>
1492 <rim:Value>%succeeds%</rim:Value>
1493 </rim:ValueList>
1494 </rim:Slot>
1495 </AdhocQueryRequest>
```

1496 **4.3.5 owl:SymmetricProperty → rim:Association Type SymmetricProperty**

1497 In OWL, if a property is symmetric, then if the pair (x,y) is an instance of the symmetric property P, then
1498 the pair (y,x) is also an instance of P [McGuinness, Harmelen]. Symmetric property is a subproperty of

1499 ObjectProperty in OWL. Consider the OWL class “WebService” and the “complements” symmetric
1500 property:

```
1501 <owl:Class rdf:ID="WebService">  
1502   <rdfs:subClassOf  
1503     rdf:resource="http://www.w3.org/2000/01/rdfschema#Resource"/>  
1504 </owl:Class>  
1505 <owl:SymmetricProperty rdf:ID="complements">  
1506   <rdfs:domain rdf:resource="#WebService"/>  
1507   <rdfs:range rdf:resource="#WebService"/>  
1508 </owl:SymmetricProperty>
```

1509 **Example owl:SymmetricProperty**

1510 Given that HotelReservationWebService complements AirReservationWebService, it is possible to
1511 deduce that AirReservationWebService complements HotelReservationWebService.

1512 owl:SymmetricProperty MUST be defined as a new type of Association in ebRIM called
1513 “SymmetricProperty”. Furthermore the following stored procedure MUST be available in the Registry to
1514 retrieve symmetric Associations of a ClassificationNode.

```
1515 CREATE PROCEDURE findSymmetricProperties($className) AS  
1516 BEGIN  
1517 SELECT A.id  
1518 FROM Association A, Name_ N, ClassificationNode C  
1519 WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-  
1520 regrep:AssociationType:SymmetricProperty' AND  
1521 C.id = N.parent AND  
1522 N.value LIKE $className AND  
1523 A.sourceObject = C.id  
1524 END;
```

1525 **Parameterized (generic) stored procedure retrieving all the Symmetric properties of a given**
1526 **classification node**

1527 The following is an example on how this stored procedure can be called:

```
1528 <AdhocQueryRequest>  
1529   <query:ResponseOption returnComposedObjects="true"  
1530   returnType="LeafClassWithRepositoryItem"/>  
1531   <rim:Slot name="urn:oasis:names:tc:ebxml-  
1532   regrep:3.0:rs:AdhocQueryRequest:queryId">  
1533     <rim:ValueList>  
1534       <rim:Value>UUID OF THE STORED QUERY</rim:Value>  
1535     </rim:ValueList>  
1536   </rim:Slot>  
1537   <rim:Slot name="$className ">  
1538     <rim:ValueList>  
1539       <rim:Value>%AirReservationServices%</rim:Value>  
1540     </rim:ValueList>  
1541   </rim:Slot>  
1542 </AdhocQueryRequest>
```

1543 **4.3.6 owl:FunctionalProperty → rim:Association Type FunctionalProperty**

1544 In OWL, if a property is a FunctionalProperty, then it has no more than one value for each individual (it
1545 may have no values for an individual) [McGuinness, Harmelen]. The range of a FunctionalProperty can be
1546 either an Object or a datatype. Consider, for example, the “hasPrice” Functional property which has a
1547 unique price:

```
1548 <owl:DatatypeProperty rdf:ID="hasPrice">  
1549   <rdfs:type rdf:resource="&owl;FunctionalProperty" />  
1550   <rdfs:domain rdf:resource="#AirReservationServices"/>  
1551   <rdfs:range  
1552   rdf:resource="http://www.w3.org/2000/10/XMLSchema/nonNegativeInteger"/>  
1553 </owl:DatatypeProperty>
```

1554 **Example owl:FunctionalProperty**

1555 ebXML RIM MUST contain a new Association Type called “FunctionalProperty” to express this semantics.
1556 Furthermore the following stored procedure MUST be available in the Registry to retrieve functional

1557 Associations of a ClassificationNode.

```
1558 CREATE PROCEDURE findFunctionalProperties($className) AS
1559 BEGIN
1560 SELECT A.id
1561 FROM Association A, Name_ N, ClassificationNode C
1562 WHERE A.associationType LIKE 'urn:oasis:names:tc:
1563 ebxml-regrep:AssociationType:FunctionalProperty' AND
1564 C.id = N.parent AND
1565 N.value LIKE $className AND
1566 A.sourceObject = C.id
1567 END;
```

1568 **Parameterized (generic) stored procedure retrieving all the Functional properties of a given**
1569 **classification node**

1570 The following is an example on how this stored procedure can be called:

```
1571 <AdhocQueryRequest>
1572 <query:ResponseOption returnComposedObjects="true"
1573 returnType="LeafClassWithRepositoryItem"/>
1574 <rim:Slot name="urn:oasis:names:tc:ebxml-
1575 regrep:3.0:rs:AdhocQueryRequest:queryId">
1576 <rim:ValueList>
1577 <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1578 </rim:ValueList>
1579 </rim:Slot>
1580 <rim:Slot name="$className ">
1581 <rim:ValueList>
1582 <rim:Value>%AirReservationServices%</rim:Value>
1583 </rim:ValueList>
1584 </rim:Slot>
1585 </AdhocQueryRequest>
```

1586 4.3.7 owl:InverseFunctionalProperty → rim:Association Type 1587 InverseFunctionalProperty

1588 In OWL, if a property is inverse functional then the inverse of the property is functional. Thus the inverse
1589 of the property has at most one value for each individual [McGuinness, Harmelen].

1590 As an example, the ObjectProperty “departsFrom” indicates that each flight originates from only one
1591 airport.

```
1592 <owl:ObjectProperty rdf:ID="departsFrom">
1593 <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
1594 <rdfs:domain rdf:resource="#Airport"/>
1595 <rdfs:range rdf:resource="#Airport"/>
1596 </owl:ObjectProperty>
```

1597 **Example owl:InverseFunctionalProperty**

1598 ebRIM MUST contain a new Association Type called “InverseFunctionalProperty” to express this
1599 semantics. Furthermore the following stored procedure MUST be available in the Registry to retrieve
1600 inverse functional Associations of a ClassificationNode.

```
1601 CREATE PROCEDURE findInverseFunctionalProperties($className) AS
1602 BEGIN
1603 SELECT A.id
1604 FROM Association A, Name_ N, ClassificationNode C
1605 WHERE A.associationType LIKE 'urn:oasis:names:tc:
1606 ebxml-regrep:AssociationType:InverseFunctionalProperty' AND
1607 C.id = N.parent AND
1608 N.value LIKE $className AND
1609 A.sourceObject = C.id
1610 END;
```

1611 **Parameterized (generic) stored procedure retrieving all the Inverse Functional properties of a**
1612 **given classification node**

1613 The following is an example on how this stored procedure can be called:

```
1614 <AdhocQueryRequest>
```

```

1615     <query:ResponseOption returnComposedObjects="true"
1616     returnType="LeafClassWithRepositoryItem"/>
1617     <rim:Slot name="urn:oasis:names:tc:ebxml-
1618     regrep:3.0:rs:AdhocQueryRequest:queryId">
1619         <rim:ValueList>
1620             <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1621         </rim:ValueList>
1622     </rim:Slot>
1623     <rim:Slot name="$className ">
1624         <rim:ValueList>
1625             <rim:Value>%AirReservationServices%</rim:Value>
1626         </rim:ValueList>
1627     </rim:Slot>
1628 </AdhocQueryRequest>

```

1629 4.4 OWL Property Restrictions in ebXML RIM

1630 An important construct of OWL is "owl:Restriction". In RDF, a property has a global scope, that is, no
1631 matter what class the property is applied to, the range of the property is the same. "owl:Restriction", on the
1632 other hand, has a local scope; restriction is applied on the property within the scope of the class where it is
1633 defined. The aim is to make ontologies more extendable and hence more reusable.

1634 For example, we may define a property "paymentMethod" for travel Web services in general and we may
1635 state that the range of this property is the class "PossiblePaymentMethods". Then, for
1636 "AirReservationServices", we may wish to restrict "paymentMethod" property to, say, "CreditCard" class as
1637 demonstrated in the following two examples:

```

1638
1639 <owl:ObjectProperty rdf:ID="paymentMethod">
1640     <rdfs:domain rdf:resource="#TravelWebService"/>
1641     <rdfs:range rdf:resource="#PossiblePaymentMethods"/>
1642 </owl:ObjectProperty >

```

1643 Example owl:ObjectProperty "paymentMethod"

```

1644
1645 <owl:Class rdf:ID="AirReservationServices">
1646     <rdfs:subClassOf>
1647         <owl:Restriction>
1648             <owl:onProperty rdf:resource="#paymentMethod"/>
1649             <owl:allValuesFrom rdf:resource=" #CreditCard"/>
1650         </owl:Restriction>
1651     </rdfs:subClassOf>
1652 </owl:Class>

```

1653 Example owl:Restriction on ObjectProperty "paymentMethod"

1654 Obviously, this serves only the purpose of reusing the "paymentMethod" property. Otherwise, a new
1655 property "paymentMethodCC" can be defined between "AirReservationServices" and the "CreditCard"
1656 classes as shown in the following:

```

1657
1658 <owl:ObjectProperty rdf:ID="paymentMethodCC">
1659     <rdfs:domain rdf:resource="#AirReservationServices"/>
1660     <rdfs:range rdf:resource="#CreditCard"/>
1661 </owl:ObjectProperty >

```

1662 Example owl:ObjectProperty "paymentMethodCC"

1663 We believe that defining a generic Association Type and and keeping track of its various restrictions in
1664 relational tables will bring considerable overhead to the system. Since an Association Type can always be
1665 defined in ebXML between any RergistryObjects, we also think that the expressive power is already there.

1666 4.5 Representing OWL Restricted Cardinality in ebXML RIM

1667 4.5.1 owl:minCardinality (only 0 or 1)

1668 In OWL, cardinality is stated on a property with respect to a particular class. If a minCardinality of 1 is
1669 stated on a property with respect to a class, then any instance of that class will be related to at least one
1670 individual by that property. This restriction is another way of saying that the property is required to have a
1671 value for all instances of the class. In OWL Lite, the only minimum cardinalities allowed are 0 or 1. A
1672 minimum cardinality of zero on a property just states (in the absence of any more specific information) that
1673 the property is optional with respect to a class [McGuinness, Harmelen].

1674 Consider for example the following OWL code which states that each instance of a “WebService” class
1675 must have at least one price:

```
1676 <owl:Class rdf:ID="WebService">  
1677   <rdfs:subClassOf>  
1678     <owl:Restriction>  
1679       <owl:onProperty rdf:resource="#hasPrice"/>  
1680       <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">  
1681 1 </owl:minCardinality>  
1682     </owl:Restriction>  
1683   </rdfs:subClassOf>  
1684 </owl:Class>
```

1685 Example owl:minCardinality

1686 In ebXML RIM, cardinalities of Association Types MUST be defined by associating a minCardinality slot
1687 with the Association Types as shown in the following example:

```
1688  
1689 <rim:Association id = "hasPriceMinCardinalityRestriction"  
1690 associationType = "urn:oasis:names:tc:ebxml-  
1691 regrep:AssociationType:ObjectProperty" sourceObject = "WebService"  
1692 targetObject = "Price">  
1693   <rim:Name>  
1694     <rim:LocalizedString value = 'hasPrice' />  
1695   </rim:Name>  
1696   <rim:Slot name="minCardinality">  
1697     <rim:ValueList>  
1698       <rim:Value>1</rim:Value>  
1699     </rim:ValueList>  
1700   </rim:Slot>  
1701 </rim:Association>
```

1702 Example Representing owl:minCardinality in ebRIM

1703 4.5.2 owl:maxCardinality (only 0 or 1)

1704 In OWL, cardinality is stated on a property with respect to a particular class. If a maxCardinality of 1 is
1705 stated on a property with respect to a class, then any instance of that class will be related to at most one
1706 individual by that property. A maxCardinality 1 restriction is sometimes called a functional or unique
1707 property. It may be useful to state that certain classes have no values for a particular property. This
1708 situation is represented by a maximum cardinality of zero on the property [McGuinness, Harmelen].

1709 Consider for example the following OWL code which states that each instance of a “WebService” class
1710 can have at most one price:

```
1711 <owl:Class rdf:ID="WebService">  
1712   <rdfs:subClassOf>  
1713     <owl:Restriction>  
1714       <owl:onProperty rdf:resource="#hasPrice"/>  
1715       <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">  
1716 1 </owl:maxCardinality>  
1717     </owl:Restriction>  
1718   </rdfs:subClassOf>  
1719 </owl:Class>
```

1720 Example owl:maxCardinality

1721 In ebXML RIM, cardinalities of Association Types MUST be defined by associating a maxCardinality slot
1722 with the Association Types as shown in the following example:

1723

```
1724 <rim:Association id = "hasPriceMaxCardinalityRestriction"  
1725 associationType = "urn:oasis:names:tc:ebxml-  
1726 regrep:AssociationType:ObjectProperty" sourceObject = "WebService"  
1727 targetObject = "Price">  
1728   <rim:Name>  
1729     <rim:LocalizedString value = 'hasPrice' />  
1730   </rim:Name>  
1731   <rim:Slot name="maxCardinality">  
1732     <rim:ValueList>  
1733       <rim:Value>1</rim:Value>  
1734     </rim:ValueList>  
1735   </rim:Slot>  
1736 </rim:Association>
```

1737 Example Representing owl:maxCardinality in ebRIM

1738 4.5.3 owl:cardinality

1739 In OWL, cardinality is provided as a convenience when it is useful to state that a property on a class has
1740 both minCardinality 0 and maxCardinality 0 or both minCardinality 1 and maxCardinality 1 [McGuinness,
1741 Harmelen].

1742 Consider for example the following OWL code which states that each instance of a "WebService" class
1743 must have exactly one price:

```
1744 <owl:Class rdf:ID="WebService">  
1745   <rdfs:subClassOf>  
1746     <owl:Restriction>  
1747       <owl:onProperty rdf:resource="#hasPrice"/>  
1748       <owl:Cardinality rdf:datatype="xsd:nonNegativeInteger"> 1  
1749     </owl:Cardinality>  
1750   </owl:Restriction>  
1751   </rdfs:subClassOf>  
1752 </owl:Class>
```

1753 Example owl:Cardinality

1754 In ebXML RIM, cardinalities of Association Types MUST be defined by associating a Cardinality slot with
1755 the Association Types as shown in the following example:

1756

```
1757 <rim:Association id = "hasPriceCardinalityRestriction"  
1758 associationType = "urn:oasis:names:tc:ebxml-  
1759 regrep:AssociationType:ObjectProperty" sourceObject = "WebService"  
1760 targetObject = "Price">  
1761   <rim:Name>  
1762     <rim:LocalizedString value = 'hasPrice' />  
1763   </rim:Name>  
1764   <rim:Slot name="cardinality">  
1765     <rim:ValueList>  
1766       <rim:Value>1</rim:Value>  
1767     </rim:ValueList>  
1768   </rim:Slot>  
1769 </rim:Association>
```

1770 Example Representing owl:Cardinality in ebRIM

1771 4.6 Representing OWL Class Intersection in ebXML RIM

1772 OWL provides the means to manipulate class extensions using basic set operators. In OWL Lite, only
1773 "owl:intersectionOf" is available which defines a class that consists of exactly all objects that belong to
1774 both of the classes. In the following example, "AirReservationServices" is defined as the intersection of

1775 "AirServices" and "ReservationServices":

1776

```
1777 <owl:Class rdf:ID="AirReservationServices">
1778   <owl:intersectionOf rdf:parseType="Collection">
1779     <owl:Class rdf:about="#AirServices" />
1780     <owl:Class rdf:about="#ReservationServices" />
1781   </owl:intersectionOf>
1782 </owl:Class>
```

1783 **Example owl:intersectionOf**

1784 In ebXML RIM "owl:intersectionOf" set operator MUST be represented as follows:

- 1785 • A new Association Type called "intersectionOf" MUST be created.
- 1786 • A new ClassificationNode to denote the intersection of the classes MUST be created. For the
1787 example, this could be "AirReservationServices" ClassificationNode.
- 1788 • Each of the intersected classes MUST be represented as members of a new RegistryPackage.
1789 For the example, the RegistryPackage should contain "AirServices" and the
1790 "RegistrationServices".
- 1791 • The new ClassificationNode denoting the intersection MUST be assigned as the sourceObject of
1792 the "intersectionOf" association. For the example, "AirReservationServices" must be the the
1793 sourceObject of the "intersectionOf" association.
- 1794 • The target class of the "intersectionOf" association MUST be set to the newly created
1795 RegistryPackage. For the example given above, the RegistryPackage containing "AirServices"
1796 and the "RegistrationServices" should be the target class of the "intersectionOf" association.

1797

```
1798 <rim:ClassificationNode id = "AirReservationServices" parent= "Service" >
1799   <rim:Name>
1800     <rim:LocalizedString value = "AirReservationServices" />
1801   </rim:Name>
1802 </rim:ClassificationNode>
1803
1804
1805 <rim:RegistryPackage id = "IntersectionOfRegistryPackage" >
1806   <rim:Name>
1807     <rim:LocalizedString value =
1808 "IntersectionOfRegistryPackage"/>
1809   </rim:Name>
1810 </rim:RegistryPackage>
1811
1812 <rim:Association id = "HasMemberRegistryPackageAssoc1"
1813 associationType = "urn:oasis:names:tc:ebxml-
1814 regrep:AssociationType:HasMember" sourceObject =
1815 "IntersectionOfRegistryPackage"
1816 targetObject = "AirServices" />
1817
1818 <rim:Association id = "HasMemberRegistryPackageAssoc2"
1819 associationType = "urn:oasis:names:tc:ebxml-
1820 regrep:AssociationType:HasMember" sourceObject =
1821 "IntersectionOfRegistryPackage"
1822 targetObject = "ReservationServices" />
1823
1824 <rim:Association id = "IntersectionOfRegistryPackageAssoc"
1825 associationType = "urn:oasis:names:tc:ebxml-
1826 regrep:AssociationType:IntersectionOf" sourceObject =
1827 "AirReservationServices"
1828 targetObject = " IntersectionOfRegistryPackage " />
1829
```

1830 **Example Defining Intersection of ClassificationNodes in ebRIM**

1831 When such a representation is used to create a complex class (a new ClassificationNode) in RIM, it
1832 becomes possible to infer that the objects (instances) classified by both of the classes

1833 (ClassificationNodes) constituting the intersection are also the instances of this complex class. The
1834 following stored procedure MUST be available in the ebXML Registry to retrieve the direct instances of the
1835 complex class and also the instances of the intersection of the classes.

1836

```
1837 CREATE PROCEDURE findInstances($className) AS
1838 BEGIN
1839 SELECT N1.value FROM Name_ N1, Service S, (
1840     SELECT A.targetObject AS id
1841     FROM RegistryPackage R, Association A
1842     WHERE R.id=A.sourceObject AND
1843           A.associationType = 'urn:oasis:names:tc:ebxml-
1844 regrep:AssociationType:HasMember' AND
1845           R.id IN (
1846             SELECT A.targetObject
1847             FROM Association A, Name_ N, ClassificationNode C
1848             WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1849 regrep:AssociationType:intersectionOf' AND
1850                   C.id = N.parent AND
1851                   N.value LIKE $className AND
1852                   A.sourceObject = C.id
1853           )
1854     ) AS T1, (
1855     SELECT A.targetObject AS id
1856     FROM RegistryPackage R, Association A
1857     WHERE R.id=A.sourceObject AND
1858           A.associationType = 'urn:oasis:names:tc:ebxml-
1859 regrep:AssociationType:HasMember' AND
1860           R.id IN (
1861             SELECT A.targetObject
1862             FROM Association A, Name_ N, ClassificationNode C
1863             WHERE A.associationType LIKE 'urn:oasis:names:tc:ebxml-
1864 regrep:AssociationType:intersectionOf' AND
1865                   C.id = N.parent AND
1866                   N.value LIKE $className AND
1867                   A.sourceObject = C.id
1868           )
1869     ) AS T2
1870 WHERE S.id IN (
1871     SELECT classifiedObject
1872     FROM Classification
1873     WHERE classificationNode=T1.id
1874     INTERSECT
1875     SELECT classifiedObject
1876     FROM Classification
1877     WHERE classificationNode=T2.id
1878     ) AND T1.id!=T2.id AND
1879     N1.parent=S.id
1880 UNION
1881 SELECT N.value
1882 FROM Service S, Name_ N
1883 WHERE S.id IN (
1884     SELECT classifiedObject
1885     FROM Classification
1886     WHERE classificationNode IN (
1887         SELECT id
1888         FROM ClassificationNode
1889         WHERE id IN (
1890             SELECT parent
1891             FROM name_
1892             WHERE value LIKE $className
1893         )
1894     )
1895     ) AND S.id=N.parent
1896 END;
```

1897

Parameterized (generic) Stored Procedure for Retrieving the instances of intersected classes

1898 The following is an example on how this stored procedure can be called:

```
1899 <AdhocQueryRequest>
1900   <query:ResponseOption returnComposedObjects="true"
1901   returnType="LeafClassWithRepositoryItem"/>
1902   <rim:Slot name="urn:oasis:names:tc:ebxml-
1903   regrep:3.0:rs:AdhocQueryRequest:queryId">
1904     <rim:ValueList>
1905       <rim:Value> UUID OF THE STORED QUERY</rim:Value>
1906     </rim:ValueList>
1907   </rim:Slot>
1908   <rim:Slot name="$className ">
1909     <rim:ValueList>
1910       <rim:Value>%AirReservationServices%</rim:Value>
1911     </rim:ValueList>
1912   </rim:Slot>
1913 </AdhocQueryRequest>
```

1914 4.7 Representing OWL Versioning in ebXML RIM

1915 4.7.1 owl:versionInfo, owl:priorVersion

1916 An owl:versionInfo statement generally has as its object a string giving information about this version, for
1917 example RCS/ CVS keywords. This statement does not contribute to the logical meaning of the ontology
1918 other than that given by the RDF(S) model theory [McGuinness, Harmelen].

1919 An owl:priorVersion statement contains a reference to another ontology. This identifies the specified
1920 ontology as a prior version of the containing ontology [McGuinness, Harmelen].

1921 In ebXML, since a RegistryObject MAY have several versions, a logical id (called lid) is also defined which
1922 is unique for different logical objects. However the lid attribute value MUST be the same for all versions of
1923 the same logical object. Therefore, almost all the underlying ebXML relational tables keep version
1924 information through "versionName" and "comment_" attributes.

1925 "owl:version" information MUST be stored in the "versionName" and "comment_" attributes of the table
1926 ClassScheme in the Registry.

1927 4.8 Representing OWL Annotation Properties in ebXML RIM

1928 4.8.1 rdfs:label

1929 rdfs:label is an instance of rdf:Property that may be used to provide a human-readable version of a
1930 resource's name [Brickley, Guha].

1931 In ebXML RIM, human readable names of resources are provided through rim:Name. rdfs:label MUST be
1932 expressed through rim:Name.

1933

```
1934 <owl:Class rdf:ID="AirReservationServices">
1935   <rdfs:label>Air Reservation Services</rdfs:label>
1936 </owl:Class>
```

1937 Example rdfs:label

1938

```
1939 <rim:ClassificationNode id = 'AirReservationServices' parent=
1940 'TravelServices' >
1941   <rim:Name>
1942     <rim:LocalizedString value = 'Air Reservation Services' />
1943   </rim:Name>
1944 </rim:ClassificationNode>
```

1945 Example rim:Name

1946 4.8.2 rdfs:comment

1947 rdfs:comment is an instance of rdf:Property that may be used to provide a human-readable description of
1948 a resource [Brickley, Guha].

1949 In ebXML RIM, this construct MUST be expressed through rim:Description.

1950

```
1951 <owl:Class rdf:ID="AirReservationServices">  
1952   <rdfs:comment>Open Travel Alliance Air Reservation Services  
1953   </rdfs:comment>  
1954 </owl:Class>
```

1955 Example rdfs:comment

1956

```
1957 <rim:ClassificationNode id = 'AirReservationServices' parent=  
1958 'TravelServices' >  
1959   <rim:Description>  
1960     <rim:LocalizedString value = 'Open Travel Alliance Air  
1961 Reservation Services' />  
1962   </rim:Description>  
1963 </rim:ClassificationNode>
```

1964 Example: rim:Description

1965 4.8.3 rdfs:seeAlso

1966 rdfs:seeAlso is an instance of rdf:Property that is used to indicate a resource that might provide additional
1967 information about the subject resource [Brickley, Guha].

1968 This construct MUST be expressed in ebXML RIM by defining an ExternalLink, called,
1969 "seeAlsoExternalLink".

1970

```
1971 <owl:Class rdf:ID="AirReservationServices">  
1972   <rdfs:seeAlso rdf:resource="http://www.opentravel.org" />  
1973 </owl:Class>
```

1974 Example rdfs:seeAlso

```
1975 <rim:ClassificationNode id = 'AirReservationServices' parent=  
1976 'TravelServices' >  
1977 </rim:ClassificationNode>  
1978  
1979 <rim:ExternalLink id = "seeAlsoExternalLink"  
1980   externalURI= "http://www.opentravel.org" >  
1981 </rim:ExternalLink>  
1982  
1983 <rim:Association id = 'seeAlsoAssociation'  
1984   associationType = 'urn:oasis:names:tc:ebxml-  
1985 regrep:AssociationType:ExternallyLinks'  
1986   sourceObject = 'AirReservationServices'  
1987   targetObject = 'seeAlsoExternalLink' />
```

1988 Example rim:seeAlsoExternalLink

1989 4.9 OWL Datatypes in ebXML RIM

1990 OWL allows the use of XML Schema datatypes to describe part of the datatype domain by simply
1991 including their URIs within an OWL ontology [McGuinness, Harmelen]. In ebXML, XML Schema datatypes
1992 SHOULD be used by providing an external link from the registry.

1993 The following example demonstrates how XML Schema datatype "integer" can be referenced through an
1994 ExternalLink called 'integer' and how to define a DatatypeProperty, namely, "hasPrice", whose target
1995 object is the defined to be ExternalLink "integer":

1996

1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009

```
<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 Association

2010 5 OWL Profile References

2011 5.1 Normative References

2012 [Brickley, Guha] Brickley, D., Guha, R.V., RDF Vocabulary Description Language 1.0: RDF Schema
2013 W3C Recommendation 10 February 2004

2014 <http://www.w3.org/TR/rdf-schema/>

2015

2016 [DAML+OIL] <http://www.daml.org/>

2017 [ebRIM] ebXML Registry Information Model version 3.0

2018 <http://docs.oasis-open.org/regrep/regrep-rim/v3.0/regrep-rim-3.0-os.pdf>

2019

2020 [ebRS] ebXML Registry Services Specification version 3.0

2021 <http://docs.oasis-open.org/regrep/regrep-rs/v3.0/regrep-rs-3.0-os.pdf>

2022 [UML] Unified Modeling Language version 1.5

2023 <http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.pdf>

2024 [ebRR-DPT] Deployment Profile Template For ebXML Registry 3.0 OASIS Specifications V_0.1.2

2025 [ebMS-DPT] Deployment Profile Template For OASIS Specification ebXML Message Service 2.0

2026 [OWL] Web Ontology Language (OWL), <http://www.w3.org/2004/OWL/>

2027 [RDF] Resource Description Framework, <http://www.w3.org/RDF/>

2028 [WSDL] WSDL Specification

2029 <http://www.w3.org/TR/wsdl>

2030 5.2 Informative References

2031 [Dogac, et. al.] Dogac A., Kabak Y., Laleci G. C. Mattocks, F. Najmi, J. Pollock

2032 Enhancing ebXML Registries to Make them OWL Aware

2033 Distributed and Parallel Databases Journal, Springer-Verlag, Vol. 18, No. 1, July 2005, pp. 9-36.

2034

2035 [IMPL] ebXML Registry 3.0 Implementations

2036 freebXML Registry: A royalty free, open source ebXML Registry Implementation

2037 <http://ebxmlrr.sourceforge.net>

2038

2039 [LeeHendler]

2040 Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web", Scientific American, May 2001.

2041

2042 [McGuinness, Harmelen] OWL Web Ontology Language Overview,

2043 W3C Recommendation 10 February 2004, <http://www.w3.org/TR/owl-features/>

2044

2045 [StaabStuder] Staab, S., Studer, R., Handbook on Ontologies, Springer, 2004.