

1 **ebXML Registry – A Tutorial**

2 **Version: 0.4**

3 **Created**

4 September 29, 2004

5 **Document Identifier**

6 regrep-tutorial-draft-04.doc

7 **Editors**

8 Farrukh Najmi, Sun Microsystems (Farrukh.Najmi@sun.com)

9 Nikola Stojanovic, RosettaNet (Nikola.Stojanovic@RosettaNet.org)

10

10	Table of Contents	
11	Table of Figures.....	4
12	1 Introduction.....	5
13	1.1 Terminology.....	5
14	1.2 Conventions	5
15	2 Overview	7
16	2.1 Overview of UML.....	7
17	2.2 Overview of Person Information Model.....	7
18	2.3 Overview of ebXML Registry Information Model.....	9
19	2.3.1 RegistryObject	11
20	2.3.2 Object Identification	11
21	2.3.3 Object Naming and Description.....	12
22	2.3.4 Object Attributes.....	12
23	Slot Attributes	12
24	2.3.5 Object Classification.....	13
25	2.3.6 Object Association.....	14
26	2.3.7 Object References To Web Content	14
27	2.3.8 Object Packaging	15
28	2.3.9 Service Description.....	15
29	3 Mapping a Domain Specific Model to ebRIM.....	16
30	3.1 Class Mapping	16
31	3.1.1 Defining a Sub-Class of ExtrinsicObject.....	16
32	3.2 Interface Mapping.....	18
33	3.3 Inheritance Mapping.....	18
34	3.3.1 Mapping of Multiple Inheritance	18
35	3.4 Method Mapping:.....	19
36	3.5 Association Mapping	19
37	3.5.1 Navigability / Direction Mapping.....	19
38	3.5.2 Role Name / Association Name Mapping	19
39	3.5.2.1 Defining a New Association Type.....	19
40	3.5.3 Aggregation Mapping	21
41	3.5.4 Composition Mapping	21
42	3.5.5 N-ary Association Mapping.....	22
43	3.5.6 XOR Associations.....	22
44	3.6 Attribute Mapping.....	23
45	3.6.1 Mapping to Identifier	24
46	Mapping to id Attribute	24
47	Mapping to Logical Id (lid) Attribute.....	24
48	Mapping to ExternalIdentifier	25
49	3.6.2 Mapping to Name and Description.....	25
50	3.6.3 Mapping to Classification	26
51	3.6.4 Mapping to ExternalLink.....	26
52	3.6.5 Direct Mapping to ebRIM Attribute	27
53	3.6.6 Mapping to Slot.....	27
54	Mapping to rim.Slot.slotName.....	27

55	Mapping to rim.Slot.slotType	28
56	Mapping to rim.Slot.values	28
57	3.7 Enumerated Type Mapping	29
58	4 Using ClassificationSchemes	30
59	4.1 Use Cases for ClassificationSchemes	30
60	4.2 Canonical ClassificationSchemes	31
61	4.3 Extending ClassificationSchemes	31
62	4.3.1 Use Cases for Extending ClassificationSchemes	31
63	4.4 Defining New ClassificationSchemes	31
64	4.4.1 Use Cases for Defining New ClassificationSchemes	31
65	5 Defining Content Management Services	32
66	5.1 Defining Content Validation Services	32
67	5.2 Defining Content Cataloging Services	32
68	6 Defining Domain Specific Queries	33
69	6.1 Identifying Common Discovery Use Cases	33
70	7 Using the Event Notification Feature	34
71	7.1 Use Cases for Event Notification	34
72	7.2 Creating Subscriptions for Events	34
73	8 Defining Access Control	37
74	8.1 Subject Role Extension	37
75	8.2 Subject Group Extension	37
76	8.2.1 Defining Custom Access Control Policies	37
77	9 Known Issues	38
78	Appendix A PIM to ebRIM: The Complete Mapping	39
79	Appendix B Tips and Tricks	40
80	B.1 Generating Unique UUIDs	40
81	B.2 Assigning Logical Id	40
82	B.3 Organizing Object in RegistryPackages	40
83	Appendix C Revision History	41
84	Appendix D References	42
85	D.1 Normative	42
86	D.2 Informative	42

88	Table of Figures	
89	Figure 1: Person Information Model: A Sample Domain Specific Model	8
90	Figure 2: Person Information Model: Inheritance View.....	9
91	Figure 3: ebXML Registry Information Model, High Level Public View.....	10
92	Figure 4: ebXML Registry Information Model, Inheritance View	11
93	Figure 5: ObjectType ClassificationScheme: Before and After Extension for	
94	LifeEvent.....	18
95	Figure 6: ObjectType ClassificationScheme: Before and After Extension For Spouse	
96	20	
97	Figure 7: Sample Association instance between a Husband and Wife pair	21
98	Figure 8: Attribute Mapping Algorithm Flowchart.....	23
99	Figure 9: Geography ClassificationScheme Example.....	30
100		

101 **1 Introduction**

102 This document is a tutorial on how to effectively customize and use an ebXML Registry
 103 for specific domains and applications. The document includes a standard methodology
 104 for mapping a domain specific information model to the ebXML Registry Information
 105 Model.

106 As more and more organization are adopting ebXML Registry standard they are faced
 107 with the recurring need to map between their domain specific information model to the
 108 ebXML Registry Information Model [ebRIM] in order to use the registry to manage their
 109 domain specific artifacts. Currently this mapping is being done in an ad hoc manner.
 110 This technical note provides the necessary guidelines, design patterns and algorithms to
 111 customize an ebXML Registry for a specific domain. Specifically, it enables a consistent
 112 mapping from domain specific information models to ebXML Registry Information
 113 Model.

114 It is not the purpose of this document to educate the reader on ebXML Registry [ebRIM],
 115 [ebRS], information modeling or the Unified Modeling Language [UML]. The reader of
 116 this document should have a good understanding of the ebXML Registry specifications
 117 and the UML 1.5 specification.

118 **1.1 Terminology**

119 The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,
 120 SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be
 121 interpreted as described in [RFC2119].

122 **1.2 Conventions**

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

- 125 • **UML Diagrams**

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

- 129 • **Identifier Placeholders**

130 Listings may contain values that reference ebXML Registry objects by their id
 131 attribute. These id values uniquely identify the objects within the ebXML Registry.
 132 For convenience and better readability, these key values are replaced by meaningful
 133 textual variables to represent such id values.

134 For example, the following placeholder refers to the unique id defined for the
 135 canonical ClassificationNode that defines the Organization ObjectType defined in
 136 [ebRIM]:

137

138

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

139

- **Constants**

140

Constant values are printed in the Courier New font always, regardless of whether they are defined by this document or a referenced document. In addition, constant values defined by this document are printed using **bold face**. The following example shows the canonical id and lid for the canonical ObjectType ClassificationScheme defined by [ebRIM]:

141

142

143

144

145

146

147

```
<rim:ClassificationScheme
  lid="urn:oasis:names:tc:ebxml-regrep:classificationScheme:ObjectType"
  id="urn:uuid:3188a449-18ac-41fb-be9f-99a1adca02cb">
```

148

1. Example Values

149

These values are represented in *italic* font. In the following, an example of a RegistryObject's name "*ACME Inc.*" is shown:

150

151

152

153

154

```
<rim:Name>
  <rim:LocalizedString value="ACME Inc." xml:lang="en-US"/>
</rim:Name>
```

155

156 **2 Overview**

157 This chapter provides an overview of ebXML Registry Information Model [ebRIM] and
158 the sample domain specific Person Information Model (PIM). The PIM is the source
159 information model for the mapping patterns defined by this document. The [ebRIM] is
160 the target for the mapping patterns defined by this document.

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

163 **2.1 Overview of UML**

164 This document will not provide an overview of UML. The reader SHOULD review UML
165 tutorials [TUT] to get a rapid understanding of [UML]. The reader MAY refer to [UML]
166 if a deeper understanding is needed.

167 Although UML defines many different types of diagrams the focus of this document is
168 the UML Class diagram. The reader SHOULD familiarize themselves with the UML
169 Class Diagram notation using [TUT] and [UML].

170 **2.2 Overview of Person Information Model**

171 Throughout this document we use a sample domain specific information model called
172 Person Information Model (PIM). This document will demonstrate the mapping
173 principals described using the PIM as source model and [ebRIM] as the target model for
174 the mapping.

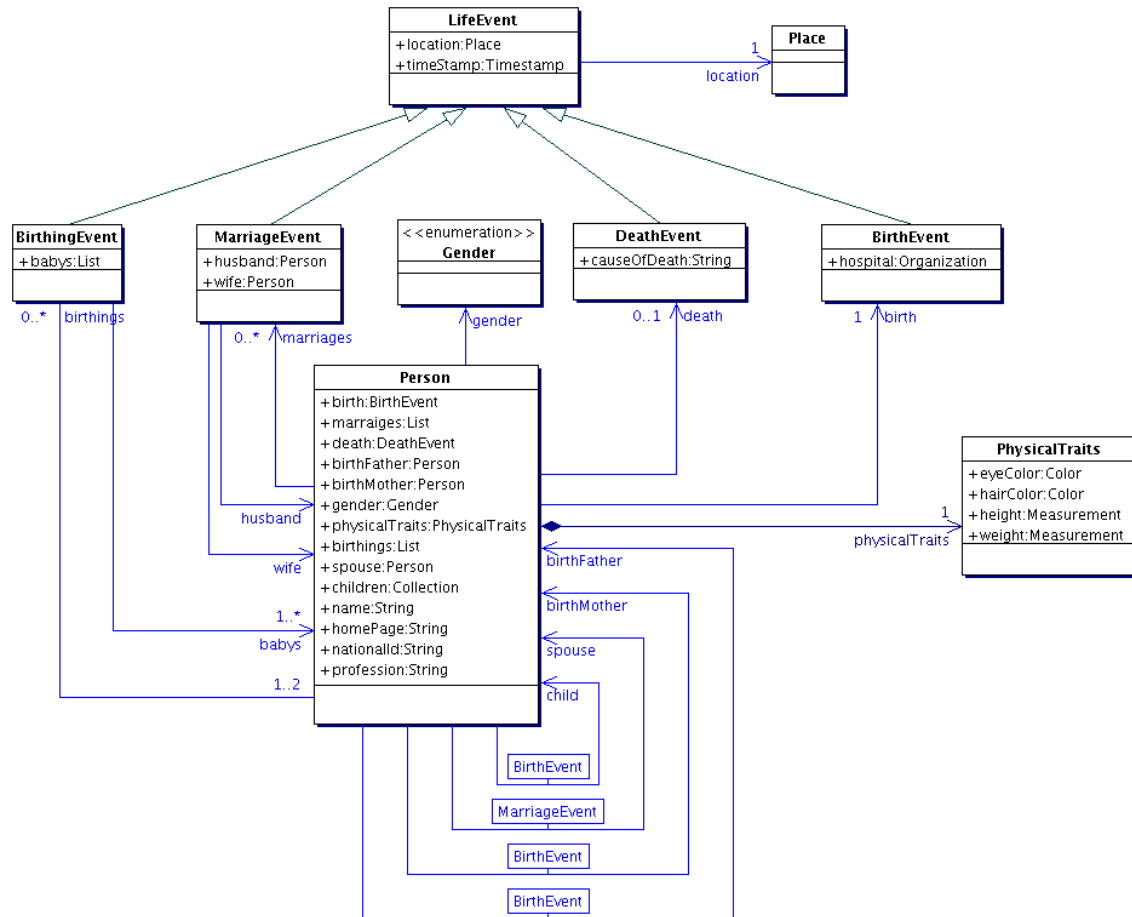


Figure 1: Person Information Model: A Sample Domain Specific Model

Figure 1 shows the UML Class diagram for the Person Information Model. The model shows that:

1. A Person has several LifeEvents:
 - o BirthEvent: Marks the birth of the associated Person
 - o MarriageEvent: Marks a marriage of the associated Person
 - o BirthingEvent: Marks a delivery of one or more babies where the associated person is a parent.
 - o DeathEvent: Marks the death of the associated Person
2. A Person has a PhysicalTraits which is a collection of various physical traits that describe the Person.
3. A Person has a birth mother and birth father which are also Person
4. A Person has children which are also Person
5. Each class MAY define various attributes as shown within the box for each class.

175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192

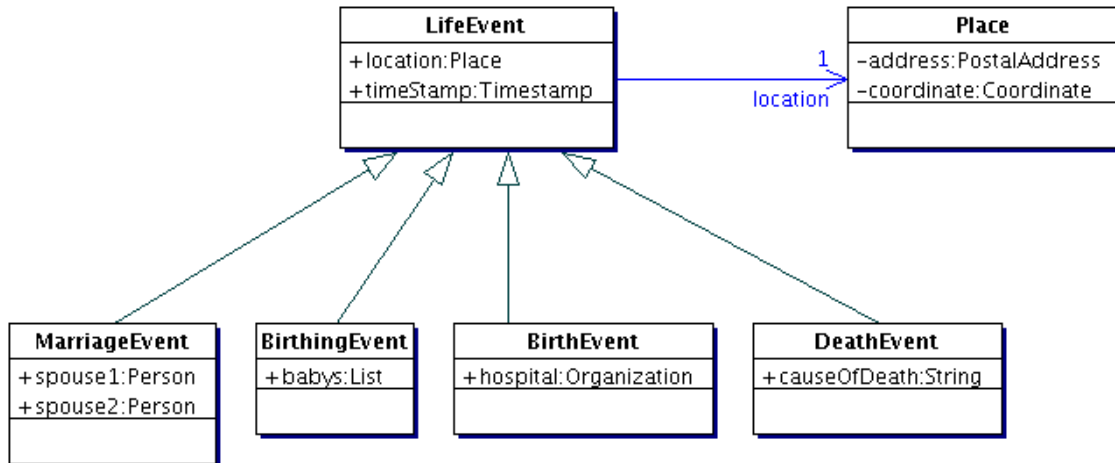


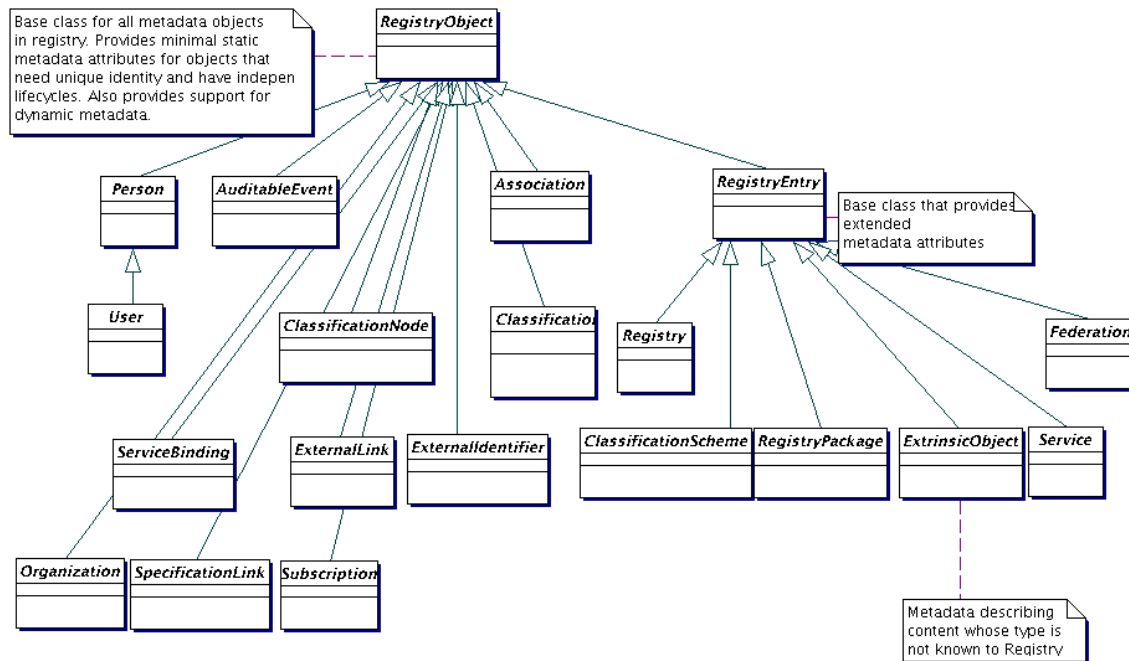
Figure 2: Person Information Model: Inheritance View

193
194
195
196
197
198

Figure 2 above shows another class diagram for the model that shows the inheritance view of the model. Here we see that the various Event classes inherit from the same LifeEvent base class and further specialize it for that specific event.

199 2.3 Overview of ebXML Registry Information Model

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



212
213

Figure 4: ebXML Registry Information Model, Inheritance View

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

215 **2.3.1 RegistryObject**

216 This is an abstract base class used by most classes in the model. It provides minimal
217 metadata for registry objects. The following sections use the Organization sub-class of
218 RegistryObject as an example to illustrate features of the model.
219

220 **2.3.2 Object Identification**

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

222
223
224

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

Listing 1: Example of id attribute

225

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

230
231
232

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

Listing 2: Example of lid Attribute

233
234

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

237

238

239

240

241

242

243

244

245

246

```

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

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

    </rim:Organization>
  
```

Listing 3: Example of ExternalIdentifier

248 **2.3.3 Object Naming and Description**

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

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

```

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

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

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

    </rim:Organization>
  
```

Listing 4: Example of Name and Description

269

270

271 **2.3.4 Object Attributes**

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

274 **Slot Attributes**

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

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

283

```

284 <rim:Organization id=${ACME_ORG_ID}
285     lid="urn:acme:ACMEOrganization" >
286
287     <rim:Slot name="urn:acme:slot:NASDAQSymbol">
288         <rim:ValueList>
289             <rim:Value>ACME</rim:Value>
290         </rim:ValueList>
291     </rim:Slot>
292
293     <rim:Name>
294         <rim:LocalizedString value="ACME Inc." xml:lang="en-US"/>
295     </rim:Name>
296     <rim:Description>
297         <rim:LocalizedString value="ACME makes Java. Provider of free Java software."
298             xml:lang="en-US"/>
299     </rim:Description>
300     <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
301         identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
302         value="ACME"/>
303     </rim:ExternalIdentifier>
304 </rim:Organization>
    
```

305

Listing 5: Example of a Dynamic Attribute Using Slot

306 **2.3.5 Object Classification**

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

311

```

312 <rim:Organization id=${ACME_ORG_ID}
313     lid="urn:acme:ACMEOrganization" >
314     <rim:Slot name="urn:acme:slot:NASDAQSymbol">
315         <rim:ValueList>
316             <rim:Value>ACME</rim:Value>
317         </rim:ValueList>
318     </rim:Slot>
319     <rim:Name>
320         <rim:LocalizedString value="ACME Inc." xml:lang="en-US"/>
321     </rim:Name>
322     <rim:Description>
323         <rim:LocalizedString value="ACME makes Java. Provider of free Java software."
324             xml:lang="en-US"/>
325     </rim:Description>
326     <rim:ExternalIdentifier id=${EXTERNAL_IDENTIFIER_ID}
327         identificationScheme=${DUNS_CLASSIFICATIONSCHEME_ID}
328         value="ACME"/>
329 </rim:ExternalIdentifier>
330
331     <!--Classify Organization as a Software Publisher using NAICS Taxonomy-->
332     <rim:Classification id=${CLASSIFICATION_ID}
333         classificationNode=${NAICS_SOFTWARE_PUBLISHER_NODE_ID}
334         classifiedObject=${ACME_ORG_ID}>
335
336 </rim:Organization>
    
```

Listing 6: Example of Object Classification

2.3.6 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 as shown in Table 1. [ebRIM] allows this list to be extensible.

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

```

351 <rim:Association
352     id=${ASSOCIATION_ID}
353     associationType=${CANONICAL_ASSOCIATION_TYPE_OFFERS_SERVICE_ID}
354     sourceObject=${ACME_ORG_ID}
355     targetObject=${ACME_SERVICE1_ID}/>
    
```

Listing 7: Example of Object Association

2.3.7 Object References To Web Content

Any RegistryObject MAY reference web content that are maintained outside the registry using association to an ExternalLink instance that contains the URL to the external web content. The following example shows the ACME Organization with an Association to an ExternalLink instance which contains the URL to ACME’s web site. The

362 associationType of the Association MUST be of type “ExternallyLinks” as defined by
 363 [ebRIM].

364

```

365 <rim:ExternalLink externalURI="http://www.acme.com"
366     id=${ACME_WEBSITE_EXTERNAL_ID}>
367 <rim:Association
368     id=${EXTERNALLYLINKS_ASSOCIATION_ID}
369     associationType=${CANONICAL_ASSOCIATION_TYPE_EXTERNALLY_LINKS_ID}
370     sourceObject=${ACME_WEBSITE_EXTERNAL_ID}
371     targetObject=${ACME_ORG_ID}/>
    
```

372

Listing 8: Example of Reference to Web Content Using ExternalLink

373 2.3.8 Object Packaging

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

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

382

```

383 <rim:RegistryPackage
384     id=${ACME_SERVICES_PACKAGE_ID}>
385   <rim:RegistryObjectList>
386     <rim:ObjectRef id=${ACME_SERVICE1_ID}
387       <rim:RegistryPackage
388         id=${ACME_PURCHASING_SERVICES_PACKAGE_ID}>
389         <rim:ObjectRef id=${ACME_PURCHASING_SERVICE1_ID}
390           <rim:ObjectRef id=${ACME_PURCHASING_SERVICE2_ID}
391         </rim:RegistryPackage>
392       <rim:RegistryPackage
393         id=${ACME_HR_SERVICES_PACKAGE_ID}>
394         <rim:ObjectRef id=${ACME_HR_SERVICE1_ID}
395           <rim:ObjectRef id=${ACME_HR_SERVICE2_ID}
396         </rim:RegistryPackage>
397     </rim:RegistryObjectList>
398   </rim:RegistryPackage>
    
```

399

Listing 9: Example of Object Packaging Using RegistryPackages

400

401 2.3.9 Service Description

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

405 **3 Mapping a Domain Specific Model to ebRIM**

406 This chapter identifies several common mapping patterns that are encountered when a
407 domain specific information model is mapped to [ebRIM]. For each such pattern we
408 define a consistent heuristic or algorithm to perform the mapping. The goal is to make it
409 easier for domain experts to utilize the ebXML Registry for their domain and to have
410 consistency across all domain-specific uses of ebXML Registry.

411 A source model may be in many different formats such as Java, XML, SQL and so on.
412 [UML] is a standard for information model description and therefore this document
413 assumes the source information model is described in UML. [UML] terminology and
414 notation is consistently used throughout this chapter and this document.

415 It should be understood that the mappings produced by applying the heuristics and
416 algorithms described in this document will be only as good as the input UML model (this
417 is the old garbage-in, garbage-out principal). A person applying these mapping patterns
418 (the mapper) MAY choose to deviate from these patterns to compensate for special
419 situations in the input UML model. Any mapping pattern not covered by this document
420 MAY be addressed in an ad hoc manner by the mapping. Suggestions for improvements
421 to the mapping should be sent to the Editors listed on the title page of this document.

422 **3.1 Class Mapping**

423 This section defines how a class in the source model is mapped to a class in [ebRIM].
424 Mapping of attributes of the source class will be discussed in section 3.6.

425

426 A class in the source model is mapped to [ebRIM] using the following algorithm:

- 427 1. **Direct Class Mapping To Rim:** First determine if there is a class in ebRIM
428 that closely matches the class in the source model. For example the Person class
429 in PIM matches closely to the Person class in [ebRIM]. Thus it is preferred that
430 the Person class in PIM is mapped to the Person class in [ebRIM].
- 431 2. **Mapping To ExtrinsicObject Sub-Class:** If no class in [ebRIM] is a good
432 match then define a new sub-class of ExtrinsicObject class in [ebRIM] and map
433 the source class to the new sub-class. See section 3.1.1 on how to define a new
434 sub-class of ExtrinsicObject. For example the various LifeEvent classes in PIM
435 SHOULD be mapped to sub-classes of ExtrinsicObject where the class names
436 match the various LifeEvent class names.

437

438 **3.1.1 Defining a Sub-Class of ExtrinsicObject**

439 This section provides the steps to define a new sub-class of ExtrinsicObject class.
440 To define a sub-class of ExtrinsicObject you MUST extend the canonical ObjectType
441 ClassificationScheme and add a new ClassificationNode as a child or descendent of the
442 canonical ClassificationNode for ExtrinsicObject in the ObjectType
443 ClassificationScheme.

444 For example to extend the ObjectType ClassificationScheme for the LifeEvent classes in
445 PIM the following ClassificationNode hierarchy MUST be submitted to the ebXML
446 Registry via a SubmitObjectsRequest.

447 Note that:

- 448 • The id attribute values SHOULD have actual id values. See Appendix A for
- 449 generating unique id values.
- 450 • The parent attribute of the LifeEvent ClassificationNode is the id of the
- 451 ExtrinsicObject ClassificationNode in the ObjectType ClassificationScheme.
- 452 • Figure 5 shows the structure of the ObjectType ClassificationScheme before and
- 453 after the extension for mapping the LifeEvent classes from PIM.

454

```

455 <!-- Add LifeEvent classes to ObjectType ClassificationScheme -->
456 <rim:ClassificationNode code="LifeEvent" id="{LIFE_EVENT_NODE_ID}"
457     parent="urn:uuid:baa2e6c8-873e-4624-8f2d-b9c7230eb4f8">
458   <rim:Name>
459     <rim:LocalizedString charset="UTF-8" value="LifeEvent"/>
460   </rim:Name>
461   <rim:ClassificationNode code="BirthEvent"
462     id="{BIRTH_EVENT_NODE_ID}">
463     <rim:Name>
464       <rim:LocalizedString charset="UTF-8" value=" BirthEvent "/>
465     </rim:Name>
466   </rim:ClassificationNode>
467   <rim:ClassificationNode code="MarriageEvent"
468     id="{MARRIAGE_EVENT_NODE_ID}">
469     <rim:Name>
470       <rim:LocalizedString charset="UTF-8" value=" MarriageEvent "/>
471     </rim:Name>
472   <rim:ClassificationNode code="BirthingEvent"
473     id="{BIRTHING_EVENT_NODE_ID}">
474     <rim:Name>
475       <rim:LocalizedString charset="UTF-8" value=" BirthingEvent "/>
476     </rim:Name>
477   </rim:ClassificationNode>
478   <rim:ClassificationNode code="DeathEvent"
479     id="{DEATH_EVENT_NODE_ID}">
480     <rim:Name>
481       <rim:LocalizedString charset="UTF-8" value=" DeathEvent "/>
482     </rim:Name>
483   </rim:ClassificationNode>
484 </rim:ClassificationNode>

```

485 **Listing 10: Example of Adding LifeEvent Classes to ObjectType ClassificationScheme**

486



Figure 5: ObjectType ClassificationScheme: Before and After Extension for LifeEvent

487
488

3.2 Interface Mapping

489

490 Interfaces are classes that only have methods and have no attributes (they may contain
491 constant attributes). They should be mapped in a manner similar to Class mapping. The
492 only difference is that Interface methods that follow the getter method design pattern
493 MAY be mapped to corresponding attributes.

494 For example, if the Person class in PIM model was an interface that had a method called
495 getAge(), then that method MAY be mapped to an age attribute in the corresponding
496 [ebRIM] class.

3.3 Inheritance Mapping

497

498 A class in the source model may have a generalization or inheritance relationship with
499 another class in the model. For example, the BirthEvent, MarriageEvent, BirthingEvent
500 and DeathEvent classes have an inheritance relationship with the LifeEvent class in PIM.
501 Such inheritance relationships SHOULD be reflected in the mapping to [ebRIM] by
502 defining a corresponding inheritance relationship among the ClassificationNodes defined
503 when extending the ObjectType scheme. This has already been illustrated in section 3.1.1
504 and Figure 5.

3.3.1 Mapping of Multiple Inheritance

505

506 A special case is “multiple inheritance” where the source model has multiple base classes
507 for the same derived class. There is no direct support for multiple inheritance in [ebRIM].
508 In case the source model has a derived class with multiple base classes, the mapping
509 SHOULD choose one base class to map as the base ClassificationNode in the ObjectType
510 ClassificationScheme. The remaining base classes SHOULD be mapped as
511 ClassificationNodes in the ObjectType ClassificationScheme and should be associated
512 with the derived class using an Association whose associationType is the id for the

513 canonical ClassificationNode “Extends” or “Implements” within the canonical
 514 AssociationType ClassificationScheme.

515 **3.4 Method Mapping:**

516 There is no support for mapping methods from a source model to [ebRIM]. Methods that
 517 follow a getter method MAY be mapped to an attribute as defined in section 3.3.

518 **3.5 Association Mapping**

519 A UML Association in the source model SHOULD be mapped to an [ebRIM]
 520 Association.

521 **3.5.1 Navigability / Direction Mapping**

522 Associations in UML MAY be directed or undirected. Associations in [ebRIM] are
 523 always implicitly directed from the sourceObject to the targetObject of an Association.
 524 Directed UML associations MUST map the Class at the arrowhead end as targetObject
 525 and the Class at the other as sourceObject. In case of Undirected UML associations the
 526 mapper MAY specify the mapping of the Classes at each end to sourceObject or
 527 targetObject using their best judgement.

528 **3.5.2 Role Name / Association Name Mapping**

529 UML defines for an association, an association name as well as two role names (one for
 530 each end of the association).

531 The role name in the UML mapping at the targetObject end of the association, if present,
 532 SHOULD be mapped to the associationType. If the role name at the targetObject end
 533 (target role name) is not present then the association name SHOULD be mapped to the
 534 associationType.

535 In addition, the target role name (or UML association name) MAY also be mapped to the
 536 Association name in ebRIM.

537 **3.5.2.1 Defining a New Association Type**

538 This section provides the steps to define a new Association Type.

539 To define a Association Type you MUST extend the canonical AssociationType
 540 ClassificationScheme and add a new ClassificationNode as a child or descendent of the
 541 AssociationType ClassificationScheme.

542 For example to extend the AssociationType ClassificationScheme for the “spouse”,
 543 “husband” and “wife” association in PIM the following ClassificationNode hierarchy
 544 SHOULD be submitted to the ebXML Registry via a SubmitObjectsRequest.

545 Note that:

- 546 • Figure 5 shows the structure of the AssociationType ClassificationScheme before and after the extension for mapping the Spouse Association Types from PIM.
- 547
- 548 • It is a good idea to organize AssociationTypes hierarchically even though the
- 549 source model may not have those semantics defined. For example it makes good

550 sense to define the “Husband” and “Wife” AssociationTypes as children of the
 551 “Spouse” AssociationType.
 552

```

553 <!-- Add Spouse, Husband, Wife to AssociationType ClassificationScheme -->
554 <rim:ClassificationNode code="Spouse" id="{ SPOUSE_NODE_ID}"
555     parent="urn:uuid:6902675f-2f18-44b8-888b-c91db8b96b4d">
556   <rim:Name>
557     <rim:LocalizedString charset="UTF-8" value="Spouse"/>
558   </rim:Name>
559   <rim:ClassificationNode code="Husband"
560     id="{ HUSBAND_NODE_ID}">
561     <rim:Name>
562       <rim:LocalizedString charset="UTF-8" value=" Husband "/>
563     </rim:Name>
564   </rim:ClassificationNode>
565   <rim:ClassificationNode code="Wife"
566     id="{ WIFE_NODE_ID}">
567     <rim:Name>
568       <rim:LocalizedString charset="UTF-8" value=" Wife "/>
569     </rim:Name>
570 </rim:ClassificationNode>
    
```

Listing 11: Example of Adding Spouse Association Types

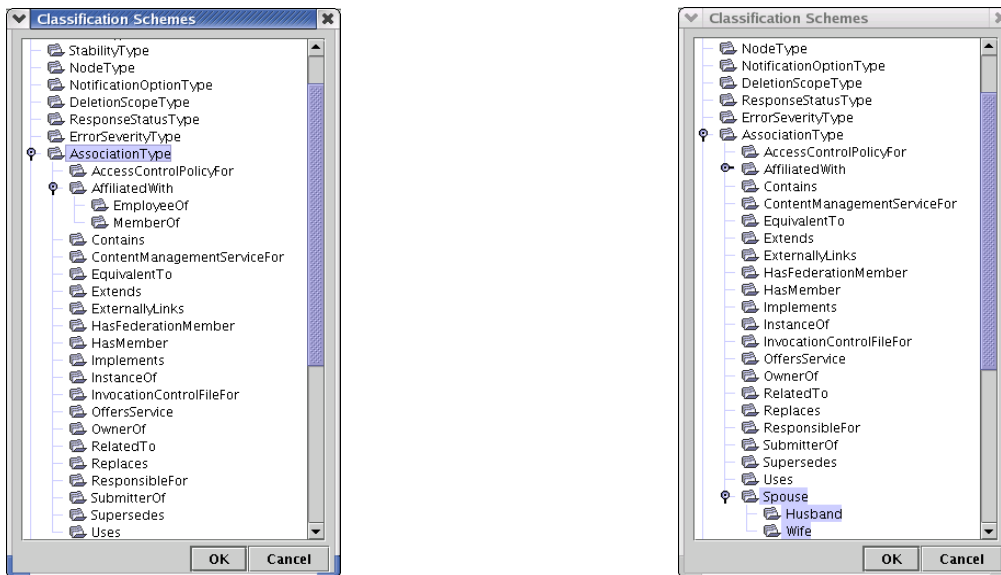


Figure 6: ObjectType ClassificationScheme: Before and After Extension For Spouse

574
 575 Figure 7 shows an example UML instance diagram to show two Associations between
 576 Person “PierreCurie” and Person “MarieCurie” in PIM. Note that the husbandToWife
 577 association has “PierreCurie” as the sourceObject and “MarieCurie” as the targetObject
 578 while the wifeToHusband associations has the two reversed.

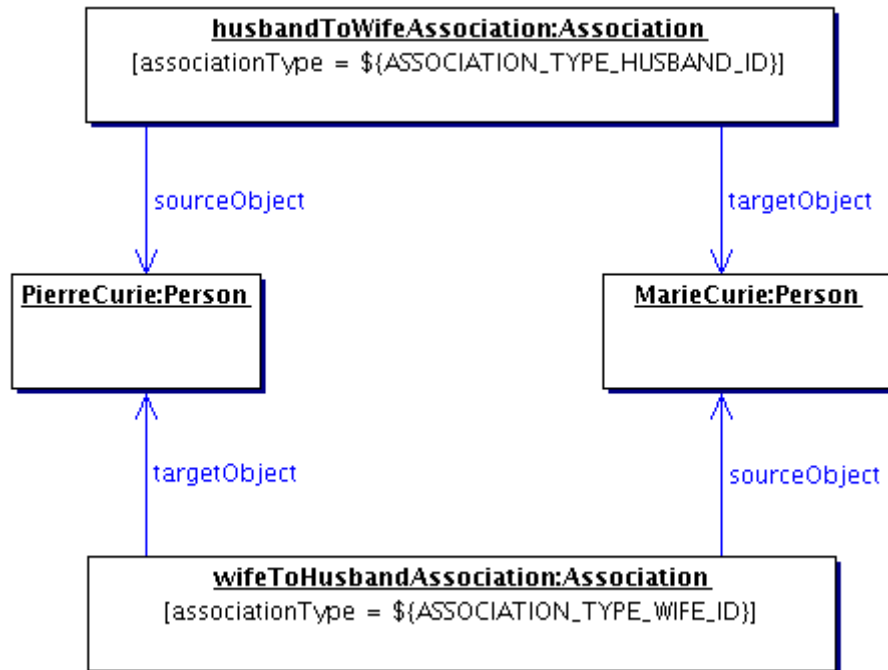


Figure 7: Sample Association instance between a Husband and Wife pair

579
580
581

3.5.3 Aggregation Mapping

A UML Aggregation maps to multiple [ebRIM] Associations in a manner consistent with earlier sections.

Give example here later??

3.5.4 Composition Mapping

When a UML Class (Container) wholly contains another class (Contained) then the UML Association between the two is called a UML Composition. The Composition Association is denoted with a filled diamond at the source end of the Association. An example of composition in PIM is where the Person class is the container while the PhysicalTraits class is the contained class.

A composition association in UML is mapped [ebRIM] as follow:

1. The container class and the contained class map to [ebRIM] as defined by section 3.1.
2. The composition Association maps to a Slot instance that is defined for the container RegistryObject.
3. The composition Slot MUST have as the value of its “name” attribute,
 - a. The target role name from the UML Association, or if that is not present
 - b. The name of the UML Association

582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600

- 601 4. The composition Slot MUST have as the value of its “slotType” attribute, the
 602 logical lid of the canonical DataType “ObjectRef”. This value is:
 603 urn:oasis:names:tc:ebxml-regrep:DataType:ObjectRef
 604 5. The composition Slot MUST have as the value of its “values” attribute, a list of
 605 String where each String MUST be the value of the id attribute of an object that is
 606 composed or contained by the container RegistryObject
 607

608 Note that the ebXML Registry does not enforce the semantics of composition
 609 Associations. Specifically, deleting a container object does not automatically delete
 610 contained objects. **Should this be added to the 3.0 specs?**
 611

612 The following example shows how the composition association between a Person
 613 instance and a PhysicalTraits instance in PIM maps to [ebRIM].
 614

```

615 <!--The ExtrinsicObject of objectType Person for Person PierreCurie -->
616 <rim:ExtrinsicObject id="{PIERRECURIE_PERSON_ID}" mimeType="text/xml"
617   objectType="{OBJECT_TYPE_PERSON_ID}">
618   <rim:Slot name="physicalTraits"
619     slotType="urn:oasis:names:tc:ebxml-regrep:DataType:ObjectRef ">
620     <rim:ValueList>
621       <rim:Value>${PIERRECURIE_PHYSICAL_TRAITS_ID}</rim:Value>
622     </rim:ValueList>
623   </rim:Slot>
624   ...
625 </rim:ExtrinsicObject>

626
627 <!--The ExtrinsicObject of objectType PhysicalTraits for Person PierreCurie -->
628 <rim:ExtrinsicObject id="{PIERRECURIE_PHYS_TRAITS_ID}"
629   mimeType="text/xml"
630   objectType="{OBJECT_TYPE_PHYS_TRAITS_ID}">
631   ...
632 </rim:ExtrinsicObject>
633
    
```

634 **Listing 12: Example of Composition of PhysicalTraits Instance Within Person Instance**

635 **3.5.5 N-ary Association Mapping**

636 UML N-ary associations involving three or more Classes is not commonly used and is
 637 not covered by this document in detail. It is suggested that RegistryPackage may be
 638 considered as a mapping for such n-ary Associations.

639 **3.5.6 XOR Associations**

640 XOR Associations as defined by UML are not commonly used in source models. XOR
 641 Associations may be mapped to [ebRIM] Associations and it MUST be the responsibility
 642 of the mapping to enforce the XOR constraints in an application specific manner.

643 **3.6 Attribute Mapping**

644 This section defines how attributes of a class in the source model are mapped to [ebRIM].
 645 Mapping of the source class to [ebRIM] has been discussed in section 3.1.

646 Figure 8 provides the flowchart for the algorithm that SHOULD be used to map attributes
 647 from the source model to [ebRIM]. Each box in right column maps to a section later in
 648 the document that describes the mapping in detail.
 649

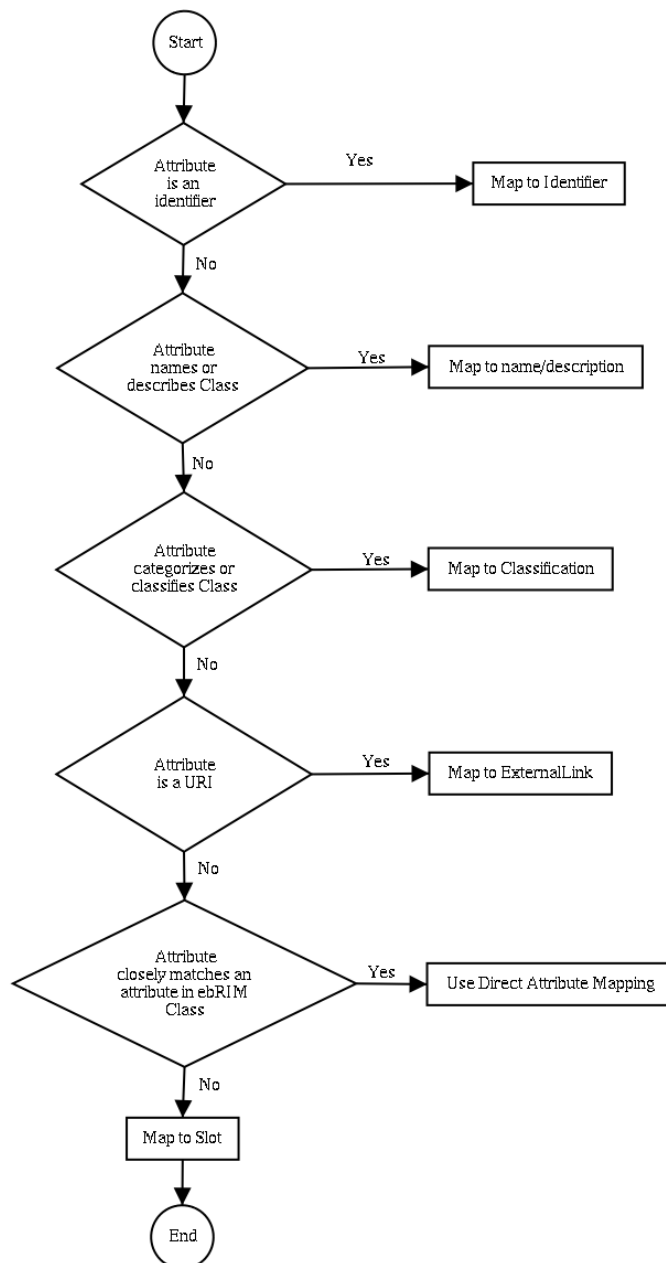


Figure 8: Attribute Mapping Algorithm Flowchart

650
 651
 652

653 **3.6.1 Mapping to Identifier**

654 Section 2.3.2 describes the various ways that a RegistryObject may be identified in
655 [ebRIM].

656 **Mapping to id Attribute**

657 If the identifier value in source model conforms to a UUID based URN as shown below,
658

```
659 urn:uuid:dafa4da3-1d92-4757-8fd8-ff2b8ce7a1bf
```

660 **Listing 13: Example of id attribute**

661 and if it provides a globally unique identifier for the source class then it **MUST** be
662 mapped to the id attribute in the target [ebRIM] class. Note that if the identifier value in
663 the source model **MUST** be the same across different versions of the same logical
664 instance of the source class then it **MUST** not be mapped to the id attribute. Instead it
665 **SHOULD** be mapped to the Logical id (lid) attribute as defined next.
666 For a detailed description of the versioning capabilities of ebXML Registry and the lid
667 attribute please see [ebRS] and [ebRIM] respectively.

668 **Mapping to Logical Id (lid) Attribute**

669 If the identifier value in the source model may be the same across all versions of an
670 instance of the class then it **SHOULD** be mapped to the lid attribute of the target class in
671 [ebRIM]. The registry requires that the lid attribute value:

- 672 • **SHOULD** be a URN
- 673 • **MUST** be unique across all logical RegistryObjects in the registry
- 674 • **MUST** be the same across all versions of the same logical RegistryObject

675
676 The lid attribute is a good way to assign a meaningful identifier to a RegistryObject. If
677 the source attribute is a human friendly identifier for the source class then it **MAY** be a
678 good candidate to be mapped to the lid attribute. Note that the source attribute value
679 need not be a URN. If it is not a URN, then the mapping **SHOULD** define a
680 deterministic algorithm for mapping the non-URN value to a URN value that meets
681 above constraints on lid attribute values.

682
683 For example, the name attribute of a Person instance in PIM **MAY** be mapped to the lid
684 attribute on the Person class in [ebRIM] using the following algorithm:

```
685 lid = "urn:pim:" + Person.name
```

686
687
688 For example the rim.Person instance for “MarieCurie” would look like:

```
689 <rim:Person id=${MARIECURIE_PERSON_ID}  
690 lid = "urn:pim:MarieCurie">  
691 ...  
692 </rim:Person>
```

694

695 Note that above example is slightly flawed because use of a person’s name in the
 696 algorithm does not guarantee that the lid would be unique since another person could
 697 have the same exact name. Also note that the urn:pim namespace MUST be registered
 698 with IANA to truly guarantee that it is a unique name space.

699 **Mapping to ExternalIdentifier**

700 If the attribute in the source model is an identifier for the source class instances but does
 701 not map to an id or lid attribute then it SHOULD be mapped to an ExternalIdentifier in
 702 [ebRIM]. The mapping MUST specify a ClassificationScheme instance that MUST be
 703 used as identificationScheme for the ExternalIdentifier.

704 For example, the nationalId attribute of the Person class in PIM may be mapped to an
 705 ExternalIdentifier that uses a ClassificationScheme named “NationalIdentifierScheme” as
 706 its identificationScheme attribute value. The mapping is responsible for defining the
 707 “NationalIdentifierScheme” ClassificationScheme as described in section 4.2.

708
 709

```

710 <rim:Person id=${MARIECURIE_PERSON_ID}
711     lid="urn:pim:MarieCurie">
712
713     <rim:ExternalIdentifier id=${NATIONAL_ID_EXTERNAL_IDENTIFIER_ID}
714         identificationScheme=${NATIONAL_ID_CLASSIFICATIONSCHEME_ID}
715         value="123-45-6789"/>
716     </rim:ExternalIdentifier>
717
718     ...
719 </rim:Person>
    
```

720 **Listing 14: Example of Mapping to ExternalIdentifier**

721

722 **3.6.2 Mapping to Name and Description**

723 If the source attribute provides a name or description for the source class instance then it
 724 SHOULD be mapped to the name or description attribute of the RegistryObject class in
 725 [ebRIM]. The rim.RegistryObject.name and rim.RegistryObject.description attributes are
 726 of type InternationalString which can contain the name and description value is multiple
 727 locales as composed LocalizedString instances. This means that the mapping SHOULD
 728 map the name and description to the appropriate locale.

729 For example the pim.Person class has a name attribute of datatype String. The mapping
 730 SHOULD map it to the rim.Person.name attribute as shown below:

731

```

732 <rim:Person id=${MARIECURIE_PERSON_ID}
733     lid="urn:pim:MarieCurie">
734
735     <rim:Name>
736         <rim:LocalizedString value="Marie Curie" xml:lang="en-US"/>
737         <rim:LocalizedString value="Marie Curie" xml:lang="fr"/>
738     </rim:Name>
739     ...
740 </rim:Person>
    
```

741 **Listing 15: Example of Mapping to name Attribute**

742 Note that the xml:lang attribute in above example SHOULD be omitted when the default
 743 locale is implied. Since a person’s name does not change with locale the above example
 744 would be better off specifying a single LocalizedString with no xml:lang attribute
 745 specified. It is showing multiple locales for illustration purposes only.

746 3.6.3 Mapping to Classification

747 If the source attribute is somehow classifying or categorizing the class instance then it
 748 SHOULD be mapped to a Classification in [ebRIM]. For an overview of Classification
 749 see section 2.3.6.

750 For example, the rim.Person.gender attribute is of datatype Gender which is an
 751 Enumeration class where the enumerated set of values are “Male”, “Female” and
 752 “Other”. The mapping MAY map pim.Person.gender to a Classification on a rim.Person
 753 instance. Since a Classification requires a ClassificationScheme, the mapping MUST
 754 specify the ClassificationScheme.

755

```

756 <rim:Person id=${MARIECURIE_PERSON_ID}
757     lid="urn:pim:MarieCurie">
758
759     <!--Classify Person as a Female using the Gender Taxonomy-->
760     <rim:Classification id=${GENDER_CLASSIFICATION_ID}
761         classificationNode=${GENDER_FEMALE_NODE_ID}
762         classifiedObject=${MARIECURIE_PERSON_ID}>
763         ...
764 </rim:Person>
    
```

Listing 16: Example of Mapping to name Attribute

765

766 Note that in above example the Gender ClassificationScheme is indirectly referenced via
 767 the ClassificationNode for “Female” within that taxonomy. See ?? for the
 768 ClassificationScheme.
 769

770 3.6.4 Mapping to ExternalLink

771 If the source attribute will always contain a URL (or a URN) then it SHOULD be
 772 mapped to an ExternalLink. For an overview of ExternalLink see section 2.3.7.

773 For example, the rim.Person.homepage attribute, if not null, always contain the URL for
 774 the Person’s homepage. It SHOULD therefore be mapped to an ExternalLink as hown
 775 below.

776 Note that an ExternalLink MUST be related to a RegistryObject using an Association
 777 instance in [ebRIM]. This allows the same ExternalLink to be shared by many
 778 RegistryObject instances.

779

```

780 <rim:Person id=${MARIECURIE_PERSON_ID}
781     lid="urn:pim:MarieCurie">
782     ...
783 </rim:Person>
784
785 <rim:ExternalLink externalURI="http://www.aip.org/history/curie/"
786     id=${MARIECURIE_WEBSITE_EXTERNAL_LINK_ID}>
787
788 <rim:Association
789     id=${MARIECURIE_HOMEPAGE_EXTERNALLYLINKS_ASSOCIATION_ID}
790     associationType=${CANONICAL_ASSOCIATION_TYPE_EXTERNALLY_LINKS_ID}
791     sourceObject=${MARIECURIE_WEBSITE_EXTERNAL_LINK_ID}
792     targetObject=${MARIECURIE_PERSON_ID}/>
793

```

Listing 17: Example of Mapping to ExternalLink

794
795

796 3.6.5 Direct Mapping to ebRIM Attribute

797 In some cases an attribute in the source model class class may closely match an attribute
798 in the [ebRIM] class. This is the most direct and preferred attribute mapping.
799 For example the Person class in PIM has an attribute “phone” (referred to as
800 pim.Person.phone) whose semantics closely match the attribute “telephoneNumbers” in
801 the Person class in [ebRIM] (referred to as rim.Person.telephoneNumbers). Thus it is
802 preferred that the pim.Person.phone attribute is mapped to rim.Person.telephoneNumbers.
803 Impedance mismatches between the source attribute data type and target attribute data
804 type MAY be handled by the mapper using domain specific knowledge. For example the
805 pim.Person.phone attribute is of datatype String while the rim.Person.telephoneNumbers
806 attribute is of datatype TelephoneNumber where TelephoneName consists of several
807 String attributes:

- 808
- 809 • “areaCode”
- 810 • “countryCode”
- 811 • “number”

812
813
814
815
816
817

Thus the mapper MUST choose which rim. TelephoneNumber attribute the
pim.Person.phone attribute maps to. As an example they MAY chose to map it the rim.
TelephoneNumber.number attribute. Alternatively, they may define a domain specific
algorithm for splitting the pim.Person.phone attribute into one, two or three components
that map to the various TelephoneNumber attributes in a deterministic manner.

818 3.6.6 Mapping to Slot

819 When all other options for mapping the source attribute are inadequate then the attribute
820 MUST be mapped to a Slot.

821 Mapping to rim.Slot.slotName

822 The source attribute name SHOULD be mapped to the rim.Slot.slotName attribute. To
823 prevent name conflicts the mapping SHOULD define a mapping algorithm that generates

824 a URN with the source attribute name as its last component. It is also suggested that the
 825 source class name be the second last component of the URN.

826 For example, the pim.Person.profession attribute SHOULD be mapped to a URN like:

827

```
828 <rim:Person id=${MARIECURIE_PERSON_ID}
829   lid="urn:pim:MarieCurie">
830   <rim:Slot name="urn:pim:Person:profession">
831     ...
832   </rim:Slot>
833   ...
834 </rim:Person>
```

835 **Listing 18: Example of Mapping pim.Person.Profession to slotName**

836

837 **Mapping to rim.Slot.slotType**

838 The rim.Slot.slotType attribute value SHOULD be defined so it conveys the datatype
 839 semantics of the Slot value. The value of the rim.Slot.slotType attribute MUST be the lid
 840 attribute value of a ClassificationNode in the canonical DataType ClassificationScheme.

841 For example, the datatype of the pim.Person.profession in PIM is String. It MUST
 842 therefore be mapped to the rim.Slot.slotType value of:

843

```
844 <rim:Person id=${MARIECURIE_PERSON_ID}
845   lid="urn:pim:MarieCurie">
846   <rim:Slot name="urn:pim:Person:profession"
847     slotType="urn:oasis:names:ebXML-regrep:DataType:String">
848     ...
849   </rim:Slot>
850   ...
851 </rim:Person>
```

852 **Listing 19: Example of Mapping DataType to slotType**

853 Note that if the datatype happens to be a Collection then the slotType should reflect the
 854 datatype of the Collection elements. In case of a heterogeneous Collection the most
 855 specific datatype from the DataType ClassificationScheme MUST be used.

856 **Mapping to rim.Slot.values**

857 The rim.Slot.values (ValueList in XML Schema) SHOULD be defined as follows:

858

- 859 • If the value is a reference (datatype/slotType is urn:oasis:names:ebXML-
 860 regrep:DataType:ObjectRef) to another RegistryObject then the value MUST be
 861 the value of the id attribute of the RegistryObject being referenced.
- 862 • If the datatype of the source attribute is not a Collection then there should only be
 863 a single “rim:Value” within the ValueList.
- 864 • If the datatype of the source attribute is a Collection then there MAY be a
 865 multiple “rim:Value” within the ValueList.

866

867 The following example shows how the pim.Person.profession attribute is specified when
 868 mapping a pim.Person instance to a rim.Person instance.

```

869 <rim:Person id=${MARIECURIE_PERSON_ID}
870   lid="urn:pim:MarieCurie">
871   <rim:Slot name="urn:pim:Person:profession"
872     slotType="urn:oasis:names:ebXML-regrep:DataType:String">
873     <rim:ValueList>
874       <rim:Value>Scientist</rim:Value>
875     </rim:ValueList>
876   </rim:Slot>
877   ...
878 </rim:Person>

```

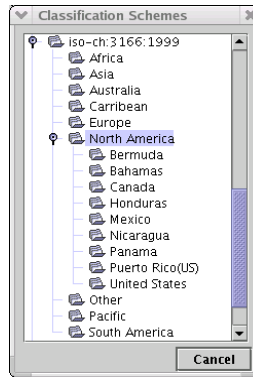
Listing 20: Example of Mapping Attribute value to Value

881 **3.7 Enumerated Type Mapping**

882 A source attribute whose datatype is an Enumeration class SHOULD be mapped to a
 883 Classification on the target RegistryObject. An example of this has been provided with
 884 the mapping of the pim.Person.gender attribute in section 3.6.3.

885 4 Using ClassificationSchemes

886 The ebXML Registry provides a powerful, simple and flexible capability to create,
 887 extend and apply taxonomies to address a wide set of use cases. A taxonomy in ebRIM is
 888 called a ClassificationScheme. The allowed values in a ClassificationScheme are
 889 represented by ClassificationNode instances within ebRIM.
 890



891
 892

Figure 9: Geography ClassificationScheme Example

893 Figure 9 shows a geography ClassificationScheme. It is a hierarchical tree structure
 894 where the root of the tree “iso-ch:3166:1999” is the name of the ClassificationScheme
 895 while the rest of the nodes in the tree are ClassificationNodes.

896 Note that most ebXML Registry implementations [IMPL] provide a GUI tool to create
 897 and manage ClassificationSchemes graphically.
 898

899 4.1 Use Cases for ClassificationSchemes

900 The following are some of the many use cases for ClassificationSchemes in an ebXML
 901 Registry:

- 902 • Used to classify RegistryObjects to facilitate discovery based upon that
 903 classification. This is the primary role of ClassificationSchemes in ebXML
 904 Registry.
- 905 • Used to define all possible values of an Enumeration class. For example, the
 906 pim.Gender class is represented in ebRIM as a Gender ClassificationScheme.
- 907 • Used to define the datatypes supported by an registry (DataType scheme).
- 908 • Used to define the Classes supported by a registry (ObjectType scheme).
- 909 • Used to define the association types supported by the registry (AssociationType
 910 scheme).
- 911 • Used to define the security roles that may be defined for users of the registry
 912 (SubjectRole scheme).
- 913 • Used to define the security groups that may be defined for users of the registry
 914 (SubjectGroup scheme).
 915

916 **4.2 Canonical ClassificationSchemes**

917 There are several ClassificationSchemes that are specified by ebRIM and required to be
 918 present in every ebXML Registry. Such standard ClassificationSchemes are referred to as
 919 “canonical” ClassificationSchemes.

920 An ebXML Registry user MAY extend existing canonical ClassificationsSchemes or add
 921 new domain specific ClassificationSchemes. However, they cannot update/delete the
 922 existing canonical ClassificationScheme or update/delete its ClassificationNodes.

923 **4.3 Extending ClassificationSchemes**

924 A registry user MAY extend an existing ClassificationScheme regardless of whether it is
 925 a canonical scheme or a user defined scheme as long as the Access Control Policies for
 926 the scheme and its nodes allow the user that privilege. The user may extend an existing
 927 scheme by submitting new ClassificationNodes to the registry that reference existing
 928 ClassificationNodes or an existing ClassificationScheme as the value of their “parent”
 929 attribute. The user SHOULD assign a logical id (lid) to all user defined
 930 ClassificationNodes for ease of identification.

931 **4.3.1 Use Cases for Extending ClassificationSchemes**

932 The following are some of the most common use cases for extending
 933 ClassificationSchemes:

- 934 • Extending the ObjectType scheme to define new Classes supported by a registry.
 935 Listing 10 shows an example of extending the ObjectType scheme.
- 936 • Extending the AssociationType scheme to define the association types supported by
 937 the registry. Listing 11 shows an example of extending the AssociationType scheme.
- 938 • Extending the SubjectRole scheme to define the security roles that may be defined for
 939 users of the registry.

940 **4.4 Defining New ClassificationSchemes**

941 A user may submit an entirely new ClassificationScheme to the registry. Often the
 942 scheme is a domain specific scheme for a specialized purpose. When mapping a domain
 943 specific model there are many situations where a new ClassificationScheme needs to be
 944 defined.

945 **4.4.1 Use Cases for Defining New ClassificationSchemes**

946

947 **5 Defining Content Management Services**

948 **5.1 Defining Content Validation Services**

949 Use of jCAM to validate XML instance docs?

950 **5.2 Defining Content Cataloging Services**

951 The ebXML Registry provides the ability for a user defined content cataloging service to
952 be configured for each ObjectType defined by the mapping. The purpose of cataloging
953 service is to selectively convert content into ebRIM compatible metadata when the
954 content is submitted. The generated metadata enables the selected content to be used as
955 parameter(s) in a domain specific parameterized query.

956 **6 Defining Domain Specific Queries**

957 The ebXML Registry provides the ability for domain specific queries to be defined as
958 parameterized stored queries within the Registry as instances of the AdhocQuery class.
959 When mapping a domain specific model one SHOULD define such domain specific
960 queries.

961 **6.1 Identifying Common Discovery Use Cases**

962 The first step in defining these domain specific queries is to identify the common use
963 cases for discovering domain specific objects in the registry using natural language.

964 For the Person Information model we identify the following sample domain specific
965 discovery use cases as likely to be commonly needed:

- 966
- 967 ○ Find Persons by:
 - 968 ○ Name
 - 969 ○ Gender
 - 970 ○ Age
 - 971 ○ # of Children
 - 972 ○ Physical trait
 - 973 ○ # of marriages
 - 974 ○ Married to specified person
 - 975 ○ Parent of specified person
 - 976 ○ Child of specified person
 - 977 ○ Ancestor of specified person
 - 978 ○ Descendent of specified person

979

980

981 **7 Using the Event Notification Feature**

982 The ebXML Registry provides the ability for a user or an automated service to create a
 983 subscription to events that match a specified criteria. Whenever an event matching the
 984 specified criteria occurs, the registry notifies the subscriber that the event transpired.
 985 A mapping of a domain specific model to ebRIM SHOULD define template
 986 Subscriptions for the typical use cases for event notification within that domain.

987 **7.1 Use Cases for Event Notification**

988 The following are some common use cases that may benefit from the event notification
 989 feature:

- 990 • A user may be using an object in the registry and may want to know when it changes.
 991 For example, they may be using an XML Schema as the schema for their XML
 992 documents. When a new version of that XML Schema is created they may wish to be
 993 notified so that they can plan the migration of their business processes to the new
 994 version of the XML Schema.
- 995 • A user may be interested in a certain type of object that does not yet exist in the
 996 registry. They may wish to be notified when such an object is published to the
 997 registry. For example, assume that a registry provides a dating service based upon
 998 PIM. Let us A person may create a subscription specifying interest in a female that
 999 has never been married before, has brown eyes, is between the age of 30 and 40 and
 1000 who is a Doctor. Whenever, a Person instance is submitted that matches this criteria,
 1001 the registry will notify the user.
- 1002 • An automated service such as a software agent may be interested in certain types of
 1003 events in the registry. For example, a state coroners office may operate a service that
 1004 wishes to be notified of deaths where the cause of death was a bullet wound. To
 1005 receive such notifications, the coroners office may create a subscription for
 1006 pim.DeathEvents where pim.DeathEvent.causeOfDeath contained the word “bullet”.

1007 **7.2 Creating Subscriptions for Events**

1008 A user may create a subscription to events of interest by submitting a Subscription object
 1009 to the registry as defined by ebRIM. The Subscription object MUST specify a selector
 1010 parameter that identifies a stored query that the registry should use to select events that
 1011 are of interest to the user for that Subscription.

```

1012 <SubmitObjectsRequest >
1013   <rim:RegistryObjectList>
1014
1015     <rim:Subscription id=${DEATH_SUBSCRIPTION_ID}
1016       selector="${SELECTOR_QUERY_ID}">
1017
1018       <!-- email address endPoint for receiving notification via email -->
1019       <rim:NotifyAction notificationOption="urn:uuid:84005f6d-419e-4138-a789-
1020 fb9fecb88f44" endPoint="mailto:farrukh.najmi@sun.com"/>
1021
1022       <!--Web Service endPoint for receiving notification via SOAP -->
1023       <rim:NotifyAction notificationOption="urn:uuid:84005f6d-419e-4138-a789-
1024 fb9fecb88f44" endPoint="urn:uuid:2a13e694-b3ae-4cda-995a-ae6b2bab3d8"/>
1025     </rim:Subscription>
1026
1027     <!-- The query used as a selector for Subscription. -->
1028     <query:SQLQuery id="${SELECTOR_QUERY_ID}">
1029       <query:QueryString>SELECT * FROM ExtrinsicObject eo WHERE
1030 eo.objectType =
1031 "${DEATH_EVENT_CLASSIFICATION_NODE_ID}"</query:QueryString>
1032     </query:SQLQuery>
1033
1034     <!-- The notification listener web service and its binding -->
1035     <rim:Service id="${DEATH_EVENT_LISTENER_SERVICE_ID}">
1036       <rim:Name>
1037         <rim:LocalizedString value="Listens for Death Events involving bullet
1038 wounds" xml:lang="en-US"/>
1039       </rim:Name>
1040
1041       <rim:ServiceBinding service=${DEATH_EVENT_LISTENER_SERVICE_ID}
1042         accessURI="http://localhost:8080/NotificationListener/notificationListener"
1043         id=${DEATH_EVENT_LISTENER_SERVICE_BINDING_ID}>
1044         <rim:Name>
1045           <rim:LocalizedString value="Death events listener web service binding"
1046             xml:lang="en-US"/>
1047         </rim:Name>
1048       </rim:ServiceBinding>
1049     </rim:Service>
1050   </rim:RegistryObjectList>
1051 </SubmitObjectsRequest>

```

Listing 21: Example of Defining a Subscription for DeathEvent

1053

1054 The above example show how a state coroner's office may create a Subscription to
 1055 DeathEvents involving bullet wounds.

1056

1057 The following notes describe the example:

- 1058 • The Subscription is submitted by sending a SubmitObjectsRequest to the registry
- 1059 as is the case when publishing any other type of RegistryObject.
- 1060 • The Subscription object is assigned a unique id, lid and optional name and
- 1061 description like any other RegistryObject.
- 1062 • The Subscription specifies the id of its selector query using the selector attribute.

- 1063 • The SubmitObjectsRequest also contains an SQLQuery object that specifies the
1064 query used to select DeathEvents. The query could be further specialized to match
1065 only those death events where the cause of death has the word “bullet” in it.
- 1066 • The subscription contains one or more NotifyActions describing how the registry
1067 should deliver notification of events matching the selector query for this
1068 subscription.
- 1069 • The subscription contains a NotifyAction that specifies an email address where
1070 the registry should send email based notification of events matching the selector
1071 query for this subscription.
- 1072 • The subscription also contains a NotifyAction that specifies the id of a
1073 ServiceBinding. This is the ServiceBinding for the automated listener service
1074 where the registry should send SOAP based based notification of events matching
1075 the selector query for this subscription.
- 1076 • The selector query and the Service / ServiceBinding MAY be submitted prior to
1077 the submission of the Subscription in a separate request.
- 1078 • Note that registry implementations [IMPL] may simplify the task of creating and
1079 managing subscriptions by providing GUI tools.
- 1080

1081 **8 Defining Access Control**

1082 The ebXML Registry provides a powerful and extensible access control feature that
 1083 makes sure that a user may only perform those actions on a RegistryObject or repository
 1084 item for which they are authorized.

1085 If you are familiar with concept of Access Control Lists (ACLs), you may think of the
 1086 registry access control feature as a similar though functionally much richer capability.
 1087 The registry provides a Role Based Access Control (RBAC) where access to objects may
 1088 be granted or denied based upon:

- 1089 • Identity of the user. An example is to grant Sally the privilege of updating the
 1090 Person instance for Marie Curie.
- 1091 • Role(s) played by user. An example is to grant anyone with role of Coroner to
 1092 update a DeathEvent instance.
- 1093 • Group(s) the user belongs to. An example is to grant anyone who belongs to the
 1094 group MarieCurieInstitute the privilege of updating the Person instance for Marie
 1095 Curie.

1096 **8.1 Subject Role Extension**

1097 The ebXML Registry defines a set of pre-defined roles in the SubjectRole scheme. A
 1098 domain specific mapping to ebRIM MAY define additional domain specific roles by
 1099 extending the SubjectRole scheme. SubjectRole scheme may be extended like any other
 1100 scheme as defined in section 4.3.

1101 **8.2 Subject Group Extension**

1102 The ebXML Registry defines a set of pre-defined roles in the SubjectGroup scheme. A
 1103 domain specific mapping to ebRIM MAY define additional domain specific groups by
 1104 extending the SubjectGroup scheme. SubjectGroup scheme may be extended like any
 1105 other scheme as defined in section 4.3.

1106 **8.2.1 Defining Custom Access Control Policies**

1107
 1108

1109 **9 Known Issues**

1110 These generic mapping patterns should be formalized via RIM artifacts and stored in the
1111 registry.

- 1112 • UML cardinality needs to be expressed generically, like for Slots, Associations,
1113 ...
- 1114 • Expanding RIM ObjectType hierarchy beyond ExtrinsicObject subtree
- 1115 • Objective criteria for when to use ObjectRefs vs. Values, like "NameAsRole"
1116 could refer to something like RoleTaxonomy instead of using value of UML role.
- 1117 • Aggregation and Composition are Association in UML. There mapping to ebRIM
1118 is inconsistent.
- 1119 • Need to give example of mapping an Association class (e.g. MarriageEvent)

1120

1121

1122 **Appendix A PIM to ebRIM: The Complete Mapping**

1123	Appendix B Tips and Tricks
1124	B.1 Generating Unique UUIDs
1125	B.2 Assigning Logical Id
1126	B.3 Organizing Object in RegistryPackages
1127	

1128

Appendix C Revision History

Rev	Date	By Whom	What
0.1	September 22, 2004	Farrukh Najmi, Nikola Stojanovic	Initial version with core mapping pattern for input from CCTS mappers.
0.2	September 23, 2004	Farrukh Najmi, Nikola Stojanovic	Minor bug fixes.
0.3	September 24, 2004	Farrukh Najmi, Nikola Stojanovic	Added some content to chapters 4-8.
0.3	September 29, 2004	Farrukh Najmi, Nikola Stojanovic	Minor fixes based upon feedback from initial reviewers.

1129

1130 **Appendix D References**

1131 **D.1 Normative**

- 1132 [ebRIM] ebXML Registry Information Model version 2.6
1133 <http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebRIM.pdf>
1134
1135 [ebRS] ebXML Registry Services Specification version 2.6
1136 <http://www.oasisopen.org/committees/regrep/documents/2.5/specs/ebRS.pdf>
- 1137 [UML] Unified Modeling Language version 1.5
1138 <http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.pdf>

1139 **D.2 Informative**

- 1140 [CMRR] Web Content Management Using OASIS ebXML Registry
1141 <http://ebxmlrr.sourceforge.net/presentations/xmlEurope2004/04-02-02.pdf>
1142 <http://ebxmlrr.sourceforge.net/presentations/xmlEurope2004/xmlEurope2004-webcm-ebxmlrr.sxi>
1143 <http://ebxmlrr.sourceforge.net/presentations/xmlEurope2004/xmlEurope2004-webcm-ebxmlrr.ppt>
- 1144 [IMPL] ebXML Registry 3.0 Implementations
1145 freebXML Registry: A royalty free, open source ebXML Registry Implementation
1146 <http://ebxmlrr.sourceforge.net>
1147 **Need other implementations listed here??**
- 1148 [TUT] UML Tutorials
1149 Borland Tutorial
1150 <http://bdn.borland.com/article/0,1410,31863,00.html>
1151 Sparx Systems UML Tutorial
1152 http://www.sparxsystems.com.au/UML_Tutorial.htm