Creating A Single Global Electronic Market

OASIS/ebXML Registry Information Model v2.5

– Committee Approved Specification

OASIS/ebXML Registry Technical Committee

June 2003

This page intentionally left blank.

1 Status of this Document

This document is an OASIS ebXML Registry Technical Committee Approved Specification - June 2003.

Distribution of this document is unlimited.

The document formatting is based on the Internet Society's Standard RFC format.

This version:

http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrim-2.5.pdf

***Latest Technical Committee Approved version*:**

http://www.oasis-open.org/committees/regrep/documents/2.1/specs/ebRIM.pdf

***Latest OASIS Approved Standard*:**

http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf

2 OASIS/ebXML Registry Technical Committee

This is an OASIS/ebXML Registry Technical Committee draft document. The following persons are members of the OASIS/ebXML Registry Technical Committee:

Kathryn Breininger, Boeing
Joseph M. Chiusano, Booz Allen Hamilton
Suresh Damodaran, Sterling Commerce
Sally Fuger, Individual Member
Peter Kacandes, Adobe Systems Incorporated
Michael Kass, NIST
Paul Macias, LMI
Matthew MacKenzie, Individual Member
Monica Martin, Sun Microsystems
Farrukh Najmi, Sun Microsystems
Sanjay Patil, IONA
Nikola Stojanovic, Individual Member
Uday Subarayan, Sun Microsystems

Contributors

The following persons contributed to the content of this document, but were not a voting member of the OASIS/ebXML Registry Technical Committee.

John Bekisz, Software AG, Inc.
Lisa Carnahan, NIST
Anne Fischer, Individual
Len Gallagher, NIST
Duane Nickull, XML Global
John Silva, Philips Medical
Sekhar Vajjhala, Sun Microsystems

Table of Contents

Table of Figures

Table of Tables

3 Introduction

3.1 Summary of Contents of Document

This document specifies the information model for the ebXML *Registry*.

A separate document, ebXML Registry Services Specification [ebRS], describes how to build *Registry Services* that provide access to the information content in the ebXML *Registry*.

3.2 General Conventions

The following conventions are used throughout this document:

UML diagrams are used as a way to concisely describe concepts. They are not intended to convey any specific *Implementation* or methodology requirements.

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

The term "*RegistryEntry*" is used to refer to an object that provides metadata about a *repository item*.

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [Bra97].

Software practitioners MAY use this document in combination with other ebXML specification documents when creating ebXML compliant software.

3.2.1 Naming Conventions

In order to enforce a consistent capitalization and naming convention in this document, "Upper Camel Case" *(UC*C) and "Lower Camel Case" *(LC*C) Capitalization styles are used in the following conventions:

- Element name is in *UCC* convention
  (example: <UpperCamelCaseElement/>)
- Attribute name is in *LCC* convention
  (example: <UpperCamelCaseElement lowerCamelCaseAttribute="whatEver"/>)
- *Class*, Interface names use UCC convention
  (examples: ClassificationNode, Versionable)
- Method name uses LCC convention
  (example: getName(), setName()).

3.3 Audience

The target audience for this specification is the community of software developers who are:

- Implementers of ebXML Registry Services
- Implementers of ebXML Registry Clients

3.4 Related Documents

The following specifications provide some background and related information to the reader:

a) ebXML Registry Services Specification [ebRS] - defines the actual Registry Services based on this information model

4 Design Objectives

4.1 Goals

The goals of this version of the specification are to:

- Communicate what information is in the Registry and how that information is organized
- Align with relevant works within other ebXML working groups
- Be able to evolve to support future ebXML Registry requirements
- Be compatible with other ebXML specifications

5 System Overview

5.1 Role of ebXML *Registry*

The registry provides a stable store where information submitted by a Submitting Organization is made persistent. Such information is used to facilitate ebXML-based Business to Business (B2B) partnerships and transactions. Submitted content may be XML schema and documents, process descriptions, ebXML Core Components, context descriptions, UML models, information about parties and even software components.

5.2 Registry Services

A set of Registry Services that provide access to registry content to clients of the registry is defined in the ebXML Registry Services Specification [ebRS]. This document does not provide details on these services but may occasionally refer to them.

5.3 What the Registry Information Model Does

The Registry Information Model provides a blueprint or high-level schema for the ebXML Registry. Its primary value is for implementers of ebXML Registries. It provides these implementers with information on the type of metadata that is stored in the registry as well as the relationships among metadata classes.

- The Registry information model:
- Defines what types of objects are stored in the Registry
- Defines how stored objects are organized in the Registry

5.4 How the Registry Information Model Works

Implementers of the ebXML Registry MAY use the information model to determine which classes to include in their registry implementation and what attributes and methods these classes may have. They MAY also use it to determine what sort of database schema their registry implementation may need.

```
[Note]The information model is meant to be
     illustrative and does not prescribe any
     specific Implementation choices.
```

5.5 Where the Registry Information Model May Be Implemented

The Registry Information Model MAY be implemented within an ebXML Registry in the form of a relational database schema, object database schema or some other physical schema. It MAY also be implemented as interfaces and classes within a registry implementation.

5.6 Conformance to an ebXML Registry

If an implementation claims conformance to this specification then it supports all required information model classes and interfaces, their attributes and their semantic definitions that are visible through the ebXML Registry Services.

6 Registry Information Model: High Level Public View

This section provides a high level public view of the objects in the *Registry*.

Figure 1 shows the high level public view of the objects in the *Registry* and their relationships as a *UML Class Diagram*. It does not show *Inheritance*, *Class* attributes or *Class* methods. The relationship links in the figure are either UML association or composition relationships (solid diamonds). In case of UML composition, instances of a class on the far side of the solid diamond are referred to as *composed objects* in the [ebRIM] and [ebRS] specifications.

The reader is again reminded that the information model is not modeling actual repository items.

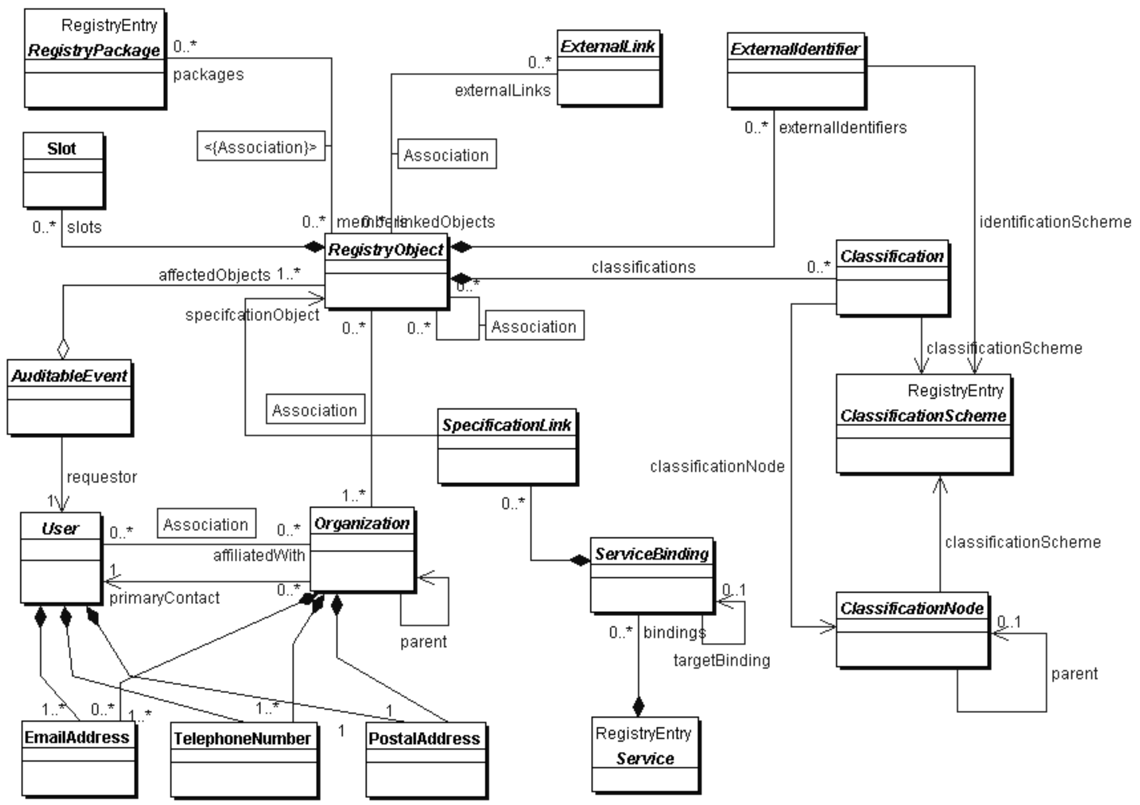

Figure 1: Information Model High Level Public View

6.1 RegistryObject

The RegistryObject class is an abstract base class used by most classes in the model. It provides minimal metadata for registry objects. It also provides methods for accessing related objects that provide additional dynamic metadata for the registry object.

6.2 RepositoryItem

The RepositoryItem class represents an object (e.g., an XML document or a DTD) that resides in a repository for storage and safekeeping. Each RepositoryItem instance is associated with a RegistryObject instance. The RegistryObject catalogs the RepositoryItem with metadata. The RepositoryItem class is not represented in the XML schema for the registry because a repository item is a mime attachment within registry requests as explained in [ebRS].

6.3 Slot

Slot instances provide a dynamic way to add arbitrary attributes to RegistryObject instances. This ability to add attributes dynamically to RegistryObject instances enables extensibility within the Registry Information Model. For example, if a company wants to add a “copyright” attribute to each RegistryObject instance that it submits, it can do so by adding a slot with name “copyright” and value containing the copyrights statement.

6.4 Association

Association instances are RegistryObject instances that are used to define many-to-many associations between objects in the information model. Associations are described in detail in section 8.

6.5 ExternalIdentifier

ExternalIdentifier instances provide additional identifier information to a RegistryObject instance, such as DUNS number, Social Security Number, or an alias name of the organization.

6.6 ExternalLink

ExternalLink instances are RegistryObject instances that contain a named URI to content external to the Registry. Unlike managed content, such external content may change or be deleted at any time without the knowledge of the Registry. A RegistryObject instance may be associated with any number of ExternalLinks.

Consider the case where a Submitting Organization submits a repository item (e.g., a DTD) and wants to associate some external content to that object (e.g., the Submitting Organization's home page). The ExternalLink enables this capability. A potential use of the ExternalLink capability may be in a GUI tool that displays the ExternalLinks to a RegistryObject. The user may click on such links and navigate to an external web page referenced by the link.

6.7 ClassificationScheme

ClassificationScheme instances are RegistryEntry instances that describe a structured way to classify or categorize RegistryObject instances. The structure of the classification scheme may be defined internal or external to the registry, resulting in a distinction between internal and external classification schemes. A very common example of a classification scheme in science is the “Classification of living things” where living things are categorized in a tree like structure. Another example is the Dewey Decimal system used in libraries to categorize books and other publications. ClassificationScheme is described in detail in section 9.

6.8 ClassificationNode

ClassificationNode instances are RegistryObject instances that are used to define tree structures under a ClassificationScheme, where each node in the tree is a ClassificationNode and the root is the ClassificationScheme. Classification trees constructed with ClassificationNodes are used to define the structure of Classification schemes or ontologies. ClassificationNode is described in detail in section 9.

6.9 Classification

Classification instances are RegistryObject instances that are used to classify other RegistryObject instances. A Classification instance identifies a ClassificationScheme instance and taxonomy value defined within the classification scheme. Classifications can be internal or external depending on whether the referenced classification scheme is internal or external. Classification is described in detail in section 9.

6.10 RegistryPackage

RegistryPackage instances are RegistryEntry instances that group logically related RegistryObject instances together.

6.11 AuditableEvent

AuditableEvent instances are RegistryObject instances that are used to provide an audit trail for RegistryObject instances. AuditableEvent is described in detail in section 11.1.

6.12 User

User instances are RegistryObject instances that are used to provide information about registered users within the Registry. User objects are used in audit trail for RegistryObject instances. User is described in detail in section 7.12.

6.13 PostalAddress

PostalAddress is a simple reusable Entity Class that defines attributes of a postal address.

6.14 EmailAddress

EmailAddress is a simple reusable Entity Class that defines attributes of an email address.

6.15 Organization

Organization instances are RegistryObject instances that provide information on organizations such as a Submitting Organization. Each Organization instance may have a reference to a parent Organization.

6.16 Service

Service instances are RegistryEntry instances that provide information on services (e.g., web services).

6.17 ServiceBinding

ServiceBinding instances are RegistryObject instances that represent technical information on a specific way to access a specific interface offered by a Service instance. A Service has a collection of ServiceBindings.

6.18 SpecificationLink

A SpecificationLink provides the linkage between a ServiceBinding and one of its technical specifications that describes how to use the service with that ServiceBinding. For example, a ServiceBinding may have a SpecificationLink instance that describes how to access the service using a technical specification in the form of a WSDL or a CORBA IDL document.

7 Registry Information Model: Detail View

This section covers the information model classes in more detail than the Public View. The detail view introduces some additional classes within the model that were not described in the public view of the information model.

Figure 2 shows the inheritance or "is a" relationships between the classes in the information model. Note that it does not show the other types of relationships, such as "has a" relationships, since they have already been shown in a previous figure. Class attributes and class methods are also not shown. Detailed description of methods and attributes of most interfaces and classes will be displayed in tabular form following the description of each class in the model.

The class Association will be covered in detail separately in section 8. The classes ClassificationScheme, Classification, and ClassificationNode will be covered in detail separately in section 9.

The reader is again reminded that the information model is not modeling actual repository items.

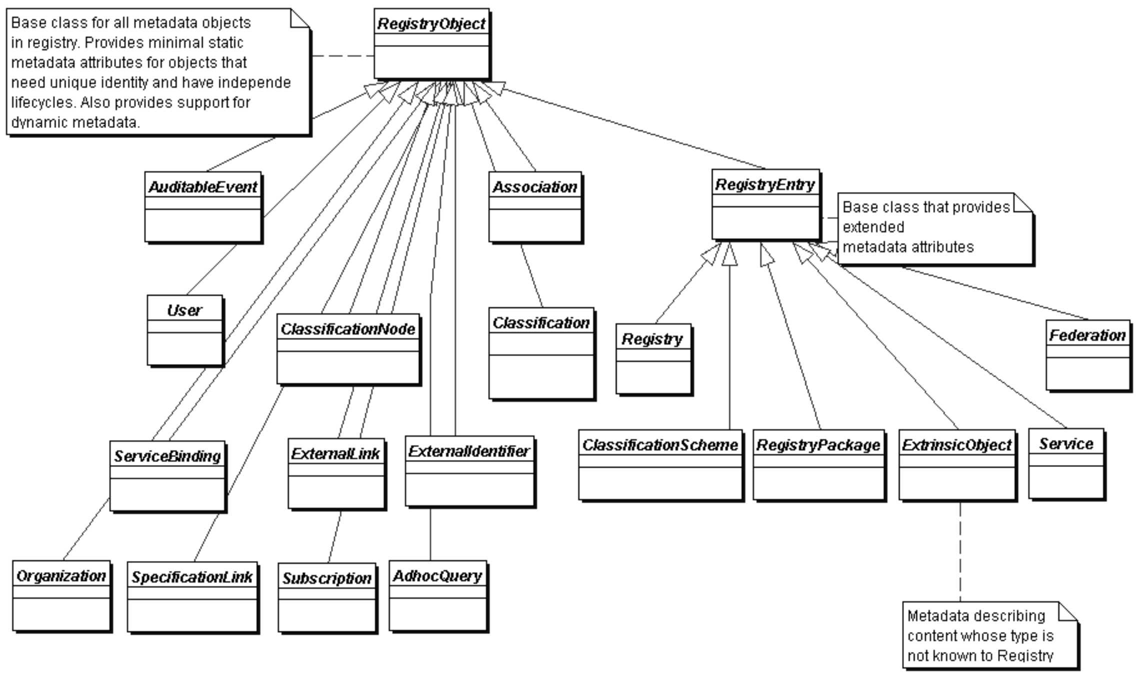
Base class for all metadata objects in registry. Provides minimal static metadata attributes for objects that need unique identity and have independe lifecycles. Also provides support for dynamic metadata.
RegistryObject
AuditableEvent
Association
RegistryEntry
Base class that provides extended metadata attributes
User
ClassificationNode
Classification
Registry
Federation
ServiceBinding
ExternalLink
ExternalIdentifier
ClassificationScheme
RegistryPackage
ExtrinsicObject
Service
Organization
SpecificationLink
Subscription
AdhocQuery
Metadata describing content whose type is not known to Registry

Figure 2: Information Model *Inheritance* View

7.1 Attribute and Methods of Information Model Classes

Information model classes are defined primarily in terms of the attributes they carry. These attributes provide information on the state of the instances of these classes. Implementations of a registry often map class attributes to attributes in an XML store or columns in a relational store.

Information model classes may also have methods defined for them. These methods provide additional behavior for the class they are defined within. Methods are currently used in mapping to Filter Query and the SQL Query capabilities defined in [ebRS].

Since the model supports inheritance between classes, it is usually the case that a class in the model inherits attributes and methods from its base classes, in addition to defining its own specialized attributes and methods.

7.2 Data Types

The following table lists the various data types used by the attributes within information model classes:

Data Type	XML Schema Data Type	Description	Length
Boolean	boolean	Used for a true or false value	
String4	string	Used for 4 character long strings	4 characters
String8	string	Used for 8 character long strings	8 characters
String16	string	Used for 16 character long strings	16 characters
String32	string	Used for 32 character long strings	32 characters
String	string	Used for unbounded Strings	unbounded
ShortName	string	A short text string	64 characters
LongName	string	A long text string	128 characters
FreeFormText	string	A very long text string for free-form text	256 characters
UUID	anyURI	A URI of the form urn:uuid:<uuid> where <uuid> Must be a DCE 128 Bit Universally unique Id.	64 characters
URI	anyURI	Used for URL and URN values	256 characters
Integer	integer	Used for integer values	4 bytes
DateTime	dateTime	Used for a timestamp value such as Date	

7.3 Object Reference Support

The information model supports the ability for an attribute in an instance of an information model class to reference a RegistryObject instance using an object reference. An object reference is modeled in this specification with the ObjectRef class.

7.3.1 Class ObjectRef

An instance of the ObjectRef class is used to reference a RegistryObject. A RegistryObject may be referenced via an ObjectRef instance regardless of its location or that of the object referring to it.

7.3.1.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
id	UUID	Yes		Client	Yes
home	URI	No		Client	Yes

7.3.1.2 Attribute id

Every ObjectRef instance must have an id attribute. The id attribute must contain the value of the id attribute of the RegistryObject being referenced.

7.3.1.3 Attribute home

Every ObjectRef instance may optionally have a home attribute specified. The home attribute if present must contain the base URI to the home registry for the referenced RegistryObject. The base URI to a registry is described by the REST interface as defined in [ebRS].

7.3.1.4 Local Vs. Remote ObjectRefs

When the home attribute is specified, and matches the base URI of a remote registry, then ObjectRef is referred to as a remote ObjectRef.

If the home attribute is null then its default value is the base URI to the current registry. When the home attribute is null or matches the base URI of the current registry, then the ObjectRef is referred to as a local ObjectRef.

7.4 Internationalization (I18N) Support

Some information model classes have String attributes that are I18N capable and may be localized into multiple native languages. Examples include the name and description attributes of the RegistryObject class in 7.5.

The information model defines the InternationalString and the LocalizedString interfaces to support I18N capable attributes within the information model classes. These classes are defined below.

7.4.1 Class InternationalString

This class is used as a replacement for the String type whenever a String attribute needs to be I18N capable. An instance of the InternationalString class composes within it Collection of LocalizedString instances, where each String is specific to a particular locale.

7.4.1.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
localized-Strings	Collection of Localized-String	No		Client	Yes

7.4.1.2 Attribute localizedStrings

Each InternationalString instance may have localizedStrings attribute that is a Collection of zero or more LocalizedString instances.

7.4.2 Class LocalizedString

This class is used as a simple wrapper class that associates a String with its locale. The class is needed in the InternationalString class where a Collection of LocalizedString instances are kept. Each LocalizedString instance has a charset and lang attribute as well as a value attribute of type String.

7.4.2.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
lang	language	No	en-US	Client	Yes
charset	String	No	UTF-8	Client	Yes
value	string	Yes		CLient	Yes

7.4.2.2 Attribute lang

Each LocalizedString instance may have a lang attribute that specifies the language used by that LocalizedString.

7.4.2.3 Attribute charset

Each LocalizedString instance may have a charset attribute that specifies the name of the character set used by that LocalizedString.

7.4.2.4 Attribute value

Each LocalizedString instance must have a value attribute that specifies the string value used by that LocalizedString.

7.5 Class RegistryObject

Direct Known Subclasses:

Association, AuditableEvent, Classification, ClassificationNode, ExternalIdentifier, ExternalLink, Organization, RegistryEntry, User, Service, ServiceBinding, SpecificationLink

---

RegistryObject provides a common base class for almost all objects in the information model. Information model Classes whose instances have a unique identity are descendants of the RegistryObject Class.

Note that Slot, PostalAddress, and a few other classes are not descendants of the RegistryObject Class because their instances do not have an independent existence and unique identity. They are always a part of some other Class's Instance (e.g., Organization has a PostalAddress).

7.5.1 Attribute Summary

The following is the first of many tables that summarize the attributes of a class. The columns in the table are described as follows:

Column	Description
Attribute	The name of the attribute
Data Type	The data type for the attribute
Required	Specifies whether the attribute is required to be specified
Default	Specifies the default value in case the attribute is omitted
Specified By	Indicates whether the attribute is specified by the client or specified by the registry. In some cases it may be both
Mutable	Specifies whether an attribute may be changed once it has been set to a certain value

Attribute	Data Type	Required	Default Value	Specified By	Mutable
classifications	Collection of Classification	No		Client	Yes
description	International-String	No		Client	Yes
externalIdentifers	Collection of ExternalLink	No		Client	Yes
id	UUID	Yes		Client or registry	No
home	URI	No		Client	Yes
name	International-String	No		Client	Yes

objectType	ObjectRef	Yes		Registry	No
slots	Collection of Slot	No		Client	Yes
status	String16	Yes		Registry	Yes

Fix sorting on all tables??

7.5.2 Attribute classifications

Each RegistryObject instance may have a Collection of zero or more Classification instances that are composed within the RegistryObject. These Classification instances classify the RegistryObject.

7.5.3 Attribute description

Each RegistryObject instance may have textual description in a human readable and user-friendly manner. This attribute is I18N capable and therefore of type InternationalString.

7.5.4 Attribute externalIdentifier

Each RegistryObject instance may have a Collection of zero or more ExternalIdentifier instances that are composed within the RegistryObject. These ExternalIdentifier instances serve as alternate identifiers for the RegistryObject.

7.5.5 Attribute id

Each RegistryObject instance must have a universally unique ID. Registry objects use the id of other RegistryObject instances for the purpose of referencing those objects.

Note that some classes in the information model do not have a need for a unique id. Such classes do not inherit from RegistryObject class. Examples include Entity classes such as TelephoneNumber, PostalAddress, EmailAddress and PersonName.

All classes derived from RegistryObject have an id that is a Universally Unique ID as defined by [UUID]. Such UUID based id attributes may be specified by the client. If the UUID based id is not specified, then it must be generated by the registry when a new RegistryObject instance is first submitted to the registry.

7.5.6 Attribute home

Each RegistryObject instance may have a home attribute. The home attribute if present, must contain the base URI to the home registry for the RegistryObject instance. The base URI to a registry is described by the REST interface as defined in [ebRS].

7.5.6.1 Local Replicas Vs. Remote Objects

When the home attribute is specified, and matches the base URI of a remote registry, then RegistryObject is referred to as a local replica of a remote RegistryObject.

If the home attribute is null then its default value is the base URI to the current registry. When the home attribute is null or matches the base URI of the current registry, then the RegistryObject is referred to as a local RegistryObject.

7.5.7 Attribute name

Each RegistryObject instance may have a human readable name. The name does not need to be unique with respect to other RegistryObject instances. This attribute is I18N capable and therefore of type InternationalString.

7.5.8 Attribute objectType

Each RegistryObject instance has an objectType attribute. The value of the objectType attribute MUST be a reference to a ClassificationNode in the canonical ObjectType ClassificationScheme as referenced in appendix A.1. A Registry MUST support the object types as defined by the ObjectType ClassificationScheme referenced in appendix A.1. The canonical ObjectType ClassificationScheme may easily be extended by adding additional ClassificationNodes to the canonical ObjectType ClassificationScheme.

The objectType for almost all objects in the information model matches the ClassificationNode that corresponds to the name of their class. For example the objectType for a Classification is a reference to the ClassificationNode with code “Classification” in the canonical ObjectType ClassificationScheme. The only exception to this rule is that the objectType for an ExtrinsicObject or an ExternalLink instance may be defined by the submitter and indicates the type of content associated with that object.

7.5.9 Attribute slots

Each RegistryObject instance may have a Collection of zero or more Slot instances that are composed within the RegistryObject. These Slot instances serve as dynamically defined attributes for the RegistryObject.

7.5.10 Attribute status

Each RegistryObject instance must have a life cycle status indicator. The status is assigned by the registry.

7.5.10.1 Pre-defined RegistryObject Status Types

The following table lists pre-defined choices for RegistryObject status attribute.

Name	Description
Submitted	Status of a RegistryObject that catalogues content that has been submitted to the registry.
Approved	Status of a RegistryObject that catalogues content that has been

	submitted to the registry and has been subsequently approved.
`Deprecated`	Status of a RegistryObject that catalogues content that has been submitted to the registry and has been subsequently deprecated.
`Undeprecated`	Status of a RegistryObject that catalogues content that has been submitted to the registry and has been deprecated and then subsequently un-deprecated.
`Withdrawn`	Status of a RegistryObject that catalogues content that has been withdrawn from the registry. A repository item has been removed but its ExtrinsicObject still exists.

7.5.11 Method Summary

In addition to its attributes, the RegistryObject class also defines the following methods. These methods are used to navigate relationship links from a RegistryObject instance to other objects.

Method Summary for RegistryObject	
`Collection`	**getAuditTrail**`()` Gets the complete audit trail of all requests that effected a state change in this object as an ordered Collection of AuditableEvent objects.
`Collection`	**getExternalLinks**`()` Gets the ExternalLinks associated with this object.
`Collection`	**getRegistryPackages**`()` Gets the RegistryPackages that this object is a member of.

7.6 Class RegistryEntry

Super Classes:
RegistryObject

Direct Known Subclasses:
ClassificationScheme, ExtrinsicObject, RegistryPackage, Service

---

RegistryEntry is a common base class for classes in the information model that require additional metadata beyond the minimal metadata provided by RegistryObject class. RegistryEntry is used as a base class for high-level coarse-grained objects in the registry. Their life cycle typically requires more management (e.g. may require approval, deprecation). They typically have relatively fewer instances but serve as a root of a composition hierarchy consisting of numerous objects that are sub-classes of RegistryObject but not RegistryEntry.

The additional metadata is described by the attributes of the RegistryEntry class below.

7.6.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
expiration	DateTime	No		Client	Yes
majorVersion	Integer	Yes	1	Registry	Yes
minorVersion	Integer	Yes	0	Registry	Yes
stability	LongName	No		Client	Yes
userVersion	ShortName	No		Client	Yes

Note that attributes inherited by RegistryEntry class from the RegistryObject class are not shown in the table above.

7.6.2 Attribute expiration

Each RegistryEntry instance may have an expirationDate. This attribute defines a time limit upon the stability indication provided by the stability attribute. Once the expirationDate has been reached the stability attribute in effect becomes STABILITY_DYNAMIC implying that the repository item can change at any time and in any manner. A null value implies that there is no expiration on stability attribute.

7.6.3 Attribute majorVersion

Each RegistryEntry instance must have a major revision number for the current version of the RegistryEntry instance. This number is assigned by the registry when the object is created and it may be changed by the registry when an object is updated.

7.6.4 Attribute minorVersion

Each RegistryEntry instance must have a minor revision number for the current version of the RegistryEntry instance. This number is assigned by the registry when the object is created and it may be changed by the registry when an object is updated.

7.6.5 Attribute stability

Each RegistryEntry instance may have a stability indicator. The stability indicator is provided by the submitter as an indication of the level of stability for the repository item.

7.6.5.1 Canonical RegistryEntry Stability Enumerations

The following table lists pre-defined choices for RegistryEntry stability attribute.

These pre-defined stability types are defined as a ClassificationScheme. While the scheme may easily be extended, a registry MAY support the stability types listed below.

Following fonts need to be fixed so consistent with RS??

Name	Description
Dynamic	Stability of a RegistryEntry that indicates that the content is dynamic and may be changed arbitrarily by submitter at any time.
DynamicCompatible	Stability of a RegistryEntry that indicates that the content is dynamic and may be changed in a backward compatible way by submitter at any time.
Static	Stability of a RegistryEntry that indicates that the content is static and will not be changed by submitter.

7.6.6 Attribute userVersion

Each RegistryEntry instance may have a userVersion. The userVersion is similar to the majorVersion-minorVersion tuple. They both provide an indication of the version of the object. The majorVersion-minorVersion tuple is provided by the registry while userVersion provides a user specified version for the object.

7.7 Class Slot

Slot instances provide a dynamic way to add arbitrary attributes to RegistryObject instances. This ability to add attributes dynamically to RegistryObject instances enables extensibility within the information model.

A RegistryObject may have 0 or more Slots. A slot is composed of a name, a slotType and a collection of values.

7.7.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
name	LongName	Yes		Client	No

slotType	LongName	No		Client	No
values	Collection of LongName	Yes		Client	No

7.7.2 Attribute name

Each Slot instance must have a name. The name is the primary means for identifying a Slot instance within a RegistryObject. Consequently, the name of a Slot instance must be locally unique within the RegistryObject instance.

7.7.3 Attribute slotType

Each Slot instance may have a slotType that allows different slots to be grouped together.

7.7.4 Attribute values

A Slot instance must have a Collection of values. The collection of values may be empty. Since a Slot represent an extensible attribute whose value may be a collection, therefore a Slot is allowed to have a collection of values rather than a single value.

7.8 Class ExtrinsicObject

Super Classes:

RegistryEntry, RegistryObject

ExtrinsicObjects provide metadata that describes submitted content whose type is not intrinsically known to the registry and therefore MUST be described by means of additional attributes (e.g., mime type).

Examples of content described by ExtrinsicObject include Collaboration Protocol Profiles [ebCPP], Business Process descriptions, and schemas.

7.8.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
isOpaque	Boolean	No	false	Client	No
mimeType	LongName	No	application/ octet-stream	Client	No

Note that attributes inherited from RegistryEntry and RegistryObject are not shown in the table above.

7.8.2 Attribute isOpaque

Each ExtrinsicObject instance may have an isOpaque attribute defined. This attribute determines whether the content catalogued by this ExtrinsicObject is opaque to (not readable by) the registry. In some situations, a Submitting Organization may submit content that is encrypted and not even readable by the registry.

7.8.3 Attribute mimeType

Each ExtrinsicObject instance may have a mimeType attribute defined. The mimeType provides information on the type of repository item catalogued by the ExtrinsicObject instance.

7.9 Class RegistryPackage

Super Classes:

RegistryEntry, RegistryObject

RegistryPackage instances allow for grouping of logically related RegistryObject instances even if individual member objects belong to different Submitting Organizations.

7.9.1 Attribute Summary

The RegistryPackage class defines no new attributes other than those that are inherited from RegistryEntry and RegistryObject base classes. The inherited attributes are not shown here.

7.9.2 Method Summary

In addition to its attributes, the RegistryPackage class also defines the following methods.

Method Summary of RegistryPackage	
`Collection`	**`getMemberObjects`**`()` Get the collection of RegistryObject instances that are members of this RegistryPackage.

7.10 Class ExternalIdentifier

Super Classes:

RegistryObject

ExternalIdentifier instances provide the additional identifier information to RegistryObject such as DUNS number, Social Security Number, or an alias name of the organization. The attribute *identificationScheme* is used to reference the identification scheme (e.g., “DUNS”, “Social Security #”), and the attribute *value* contains the actual information (e.g., the DUNS number, the social security number). Each RegistryObject may contain 0 or more ExternalIdentifier instances.

7.10.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
identificationScheme	ObjectRef	Yes		Client	Yes
registryObject	ObjectRef	Yes		Client	No
value	LongName	Yes		Client	Yes

Note that attributes inherited from the base classes of this class are not shown.

7.10.2 Attribute identificationScheme

Each ExternalIdentifier instance must have an identificationScheme attribute that references a ClassificationScheme. This ClassificationScheme defines the namespace within which an identifier is defined using the value attribute for the RegistryObject referenced by the RegistryObject attribute.

7.10.3 Attribute registryObject

Each ExternalIdentifier instance must have a *registryObject* attribute that references the parent RegistryObject for which this is an ExternalIdentifier.

7.10.4 Attribute value

Each ExternalIdentifier instance must have a *value* attribute that provides the identifier value for this ExternalIdentifier (e.g., the actual social security number).

7.11 Class ExternalLink

Super Classes:

RegistryObject

ExternalLinks use URIs to associate content in the registry with content that may reside outside the registry. For example, an organization submitting a DTD could use an ExternalLink to associate the DTD with the organization's home page.

7.11.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
externalURI	URI	Yes		Client	Yes

7.11.2 Attribute externalURI

Each ExternalLink instance must have an externalURI attribute defined. The externalURI attribute provides a URI to the external resource pointed to by this ExternalLink instance. If the URI is a URL then a registry must validate the URL to be resolvable at the time of submission before accepting an ExternalLink submission to the registry.

7.11.3 Method Summary

In addition to its attributes, the ExternalLink class also defines the following methods.

Method Summary of ExternalLink	
`Collection`	**`getLinkedObjects`**`()` Gets the collection of RegistryObjects that are linked by this ExternalLink to content outside the registry.

7.12 Class User

Super Classes:

RegistryObject

User instances represent users that have registered with a registry. User instances are also used in an AuditableEvent to keep track of the identity of the requestor that sent the request that generated the AuditableEvent.

7.12.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
addresses	Collection of PostalAddress	Yes		Client	Yes
emailAddresses	Collection of EmailAddress	Yes		Client	Yes
personName	PersonName	Yes		Client	No
telephoneNumbers	Collection of	Yes		Client	Yes

	TelephoneNumber				
url	URI	No		Client	Yes

7.12.2 Attribute addresses

Each User instance has an attribute addresses that is a Collection of PostalAddress instances. Each PostalAddress provides an postal address for that user. A User must have at least one postal address.

7.12.3 Attribute emailAddresses

Each User instance has an attribute emailAddresses that is a Collection of EmailAddress instances. Each EmailAddress provides an email address for that user. A User must have at least one email address.

7.12.4 Attribute personName

Each User instance must have a *personName* attribute that provides the name for that user.

7.12.5 Attribute telephoneNumbers

Each User instance must have a *telephoneNumbers* attribute that contains the Collection of TelephoneNumber instances defined for that user. A User must have at least one TelephoneNumber.

7.12.6 Attribute *url*

Italicise all attributes in headings and para as done in *url* above??

Each User instance may have a *url* attribute that provides the URL address for the web page associated with that user.

7.12.7 Associating Users With Organizations

A user may be affiliated with zero or more organizations. Each such affiliation is modeled in ebRIM using an Association instance between a User instance and an Organization instance. The associationType in such cases should be either the canonical "AffiliatedWith" associationType or a ClassificationNode that is a descendant of the ClassificationNode representing the canonical "AffiliatedWith" associationType.

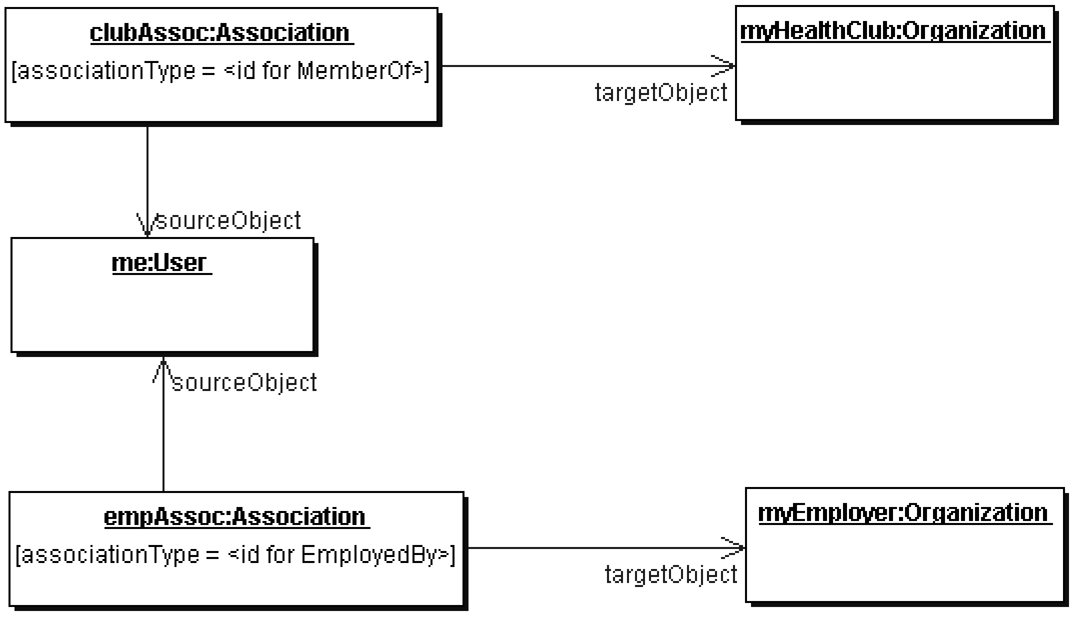

Figure 3: User Affiliation With Organization Instance Diagram

7.13 Class Organization

Super Classes:

RegistryObject

Organization instances provide information on organizations such as a Submitting Organization. Each Organization instance may have a reference to a parent Organization.

7.13.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
Address	PostalAddress	Yes		Client	Yes
emailAddresses	Collection of EmailAddress	No		Client	Yes
parent	ObjectRef	No		Client	Yes
primaryContact	ObjectRef	Yes		Client	No
telephoneNumbers	Collection of TelephoneNumber	Yes		Client	Yes

7.13.2 Attribute address

Each Organization instance must have an *address* attribute that provides the postal address for that organization.

7.13.3 Attribute emailAddresses

Each Organization instance may have an attribute *emailAddresses* that is a Collection of EmailAddress instances. Each EmailAddress provides an email address for that Organization.

7.13.4 Attribute parent

Each Organization instance may have a *parent* attribute that references the parent Organization instance, if any, for that organization.

7.13.5 Attribute primaryContact

Each Organization instance must have a *primaryContact* attribute that references the User instance for the user that is the primary contact for that organization.

7.13.6 Attribute telephoneNumbers

Each Organization instance must have a *telephoneNumbers* attribute that contains the Collection of TelephoneNumber instances defined for that organization. An Organization must have at least one telephone number.

7.13.7 Associating Organizations With RegistryObjects

An organization may be associated with zero or more RegistryObject instances. Each such association is modeled in ebRIM using an Association instance between an Organization instance and a RegistryObject instance. The associationType in such cases may be (but not restricted to) either the canonical “SubmitterOf” associationType or the canonical “ResponsibleFor” associationType. The “SubmitterOf” associationType indicates the organization that submitted the RegistryObject (via a User). The “ResponsibleFor” associationType indicates the organization that is designated as the organization responsible for the ongoing maintenance of the RegistryObject.

Association between Organizations and RegistryObjects do not entitle any special privileges for the organizations with respect to the RegistryObject. Such privileges are defined by the Access Control Policies defined for the RegistryObject as described in chapter 13.

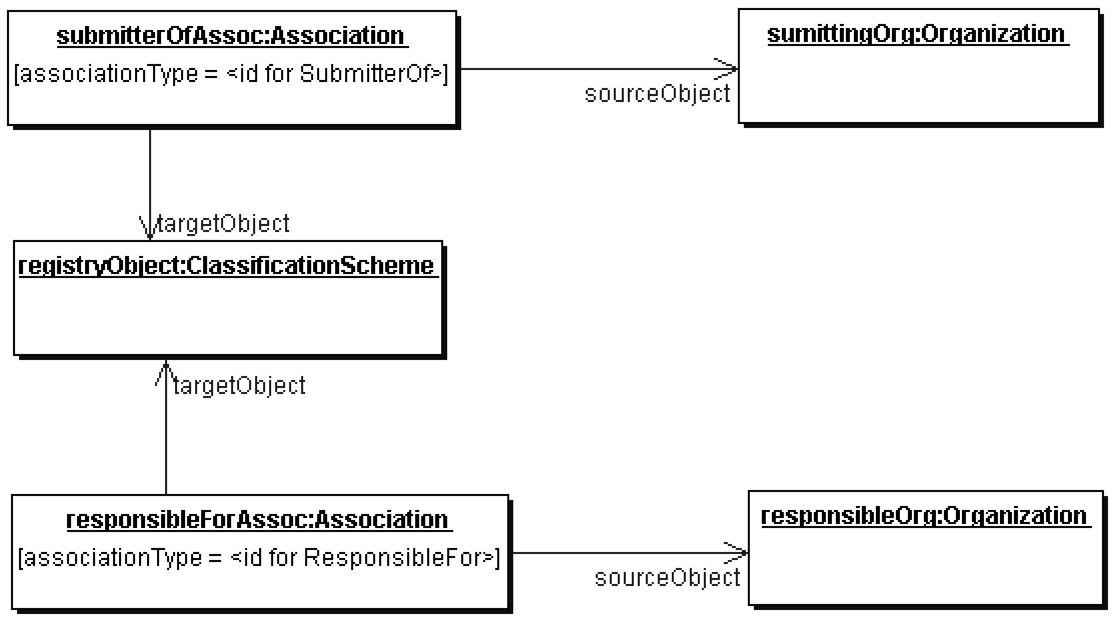
submitterOfAssoc:Association
[associationType = <id for SubmitterOf>]
sourceObject
sumittingOrg:Organization
targetObject
registryObject:ClassificationScheme
targetObject
responsibleForAssoc:Association
[associationType = <id for ResponsibleFor>]
sourceObject
responsibleOrg:Organization

Figure 4: Organization to RegistryObject Association Instance Diagram

7.14 Class PostalAddress

PostalAddress is a simple reusable Entity Class that defines attributes of a postal address.

7.14.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
city	ShortName	No		Client	Yes
country	ShortName	No		Client	Yes
postalCode	ShortName	No		Client	Yes
state	ShortName	No		Client	Yes
street	ShortName	No		Client	Yes
streetNumber	String32	No		Client	Yes

7.14.2 Attribute city

Each PostalAddress may have a *city* attribute identifying the city for that address.

7.14.3 Attribute country

Each PostalAddress may have a *country* attribute identifying the country for that address.

7.14.4 Attribute postalCode

Each PostalAddress may have a *postalCode* attribute identifying the postal code (e.g., zip code) for that address.

7.14.5 Attribute state

Each PostalAddress may have a *state* attribute identifying the state, province or region for that address.

7.14.6 Attribute street

Each PostalAddress may have a *street* attribute identifying the street name for that address.

7.14.7 Attribute streetNumber

Each PostalAddress may have a *streetNumber* attribute identifying the street number (e.g., 65) for the street address.

7.14.8 Method Summary

In addition to its attributes, the PostalAddress class also defines the following methods.

Method Summary of ExternalLink	
`Collection`	**`getSlots`**`()` Gets the collection of Slots for this object. Each PostalAddress may have multiple Slot instances where a Slot is a dynamically defined attribute. The use of Slots allows the client to extend PostalAddress class by defining additional dynamic attributes using slots to handle locale specific needs.

7.15 Class TelephoneNumber

A simple reusable Entity Class that defines attributes of a telephone number.

7.15.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
areaCode	String8	No		Client	Yes
countryCode	String8	No		Client	Yes
extension	String8	No		Client	Yes
number	String16	No		Client	Yes

phoneType	ObjectRef	No		Client	Yes
url	URI	No		Client	Yes

7.15.2 Attribute areaCode

Each TelephoneNumber instance may have an *areaCode* attribute that provides the area code for that telephone number.

7.15.3 Attribute countryCode

Each TelephoneNumber instance may have a *countryCode* attribute that provides the country code for that telephone number.

7.15.4 Attribute extension

Each TelephoneNumber instance may have an *extension* attribute that provides the extension number, if any, for that telephone number.

7.15.5 Attribute number

Each TelephoneNumber instance may have a *number* attribute that provides the local number (without area code, country code and extension) for that telephone number.

7.15.6 Attribute phoneType

Each TelephoneNumber instance may have *phoneType* attribute that provides the type for the TelephoneNumber. The value of the phoneType attribute MUST be a reference to a ClassificationNode in the canonical PhoneType ClassificationScheme as referenced in appendix A.3.

7.16 Class EmailAddress

A simple reusable Entity Class that defines attributes of an email address.

7.16.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
address	ShortName	Yes		Client	Yes
type	ObjectRef	No		Client	Yes

7.16.2 Attribute address

Each EmailAddress instance must have an *address* attribute that provides the actual email address.

7.16.3 Attribute type

Each EmailAddress instance may have a *type* attribute that provides the type for that email address. The value of the type attribute MUST be a reference to a ClassificationNode in the canonical EmailType ClassificationScheme as referenced in appendix A.4.

7.17 Class PersonName

A simple Entity Class for a person’s name.

7.17.1 Attribute Summary

Attribute	**Data Type**	**Required**	**Default Value**	**Specified By**	**Mutable**
firstName	ShortName	No		Client	Yes
lastName	ShortName	No		Client	Yes
middleName	ShortName	No		Client	Yes

7.17.2 Attribute firstName

Each PersonName may have a *firstName* attribute that is the first name of the person.

7.17.3 Attribute lastName

Each PersonName may have a *lastName* attribute that is the last name of the person.

7.17.4 Attribute middleName

Each PersonName may have a *middleName* attribute that is the middle name of the person.

8 Association Information Model

A RegistryObject instance may be associated with zero or more RegistryObject instances. The information model defines an Association class, an instance of which may be used to associate any two RegistryObject instances.

8.1 Example of an Association

One example of such an association is between two ClassificationScheme instances, where one ClassificationScheme supersedes the other ClassificationScheme as shown in Figure 5. This may be the case when a new version of a ClassificationScheme is submitted.

In Figure 5, we see how an Association is defined between a new version of the NAICS ClassificationScheme and an older version of the NAICS ClassificationScheme.

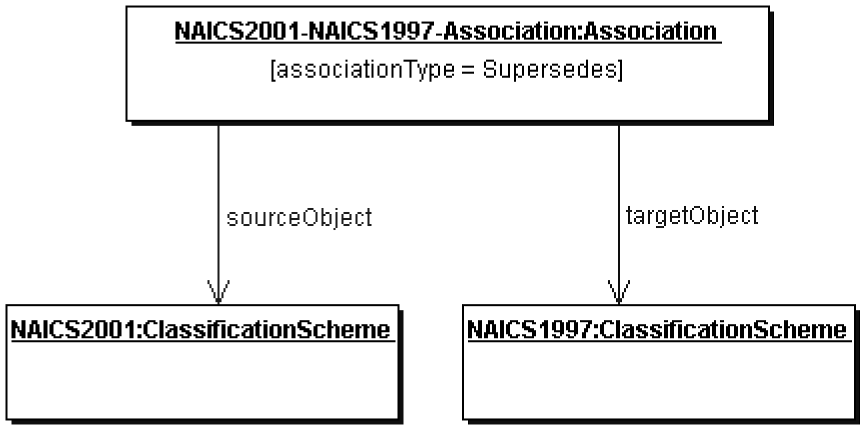
NAICS2001-NAICS1997-Association:Association
[associationType = Supersedes]
sourceObject
targetObject
NAICS2001:ClassificationScheme
NAICS1997:ClassificationScheme

Figure 5: Example of RegistryObject Association

8.2 Source and Target Objects

An Association instance represents an association between a source RegistryObject and a target RegistryObject. These are referred to as *sourceObject* and *targetObject* for the Association instance. It is important which object is the sourceObject and which is the targetObject as it determines the directional semantics of an Association.

In the example in Figure 5, it is important to make the newer version of NAICS ClassificationScheme be the sourceObject and the older version of NAICS be the targetObject because the associationType implies that the sourceObject supersedes the targetObject (and not the other way around).

8.3 Association Types

Each Association must have an associationType attribute that identifies the type of that association.

8.4 Intramural Association

A common use case for the Association class is when a User “u” creates an Association “a” between two RegistryObjects “o1” and “o2” where Association “a” and RegistryObjects “o1” and “o2” are objects that were created by the same User “u.” This is the simplest use case, where the Association is between two objects that are owned by the same User that is defining the Association. Such Associations are referred to as intramural Associations.

Figure 6 below, extends the previous example in Figure 5 for the intramural Association case.

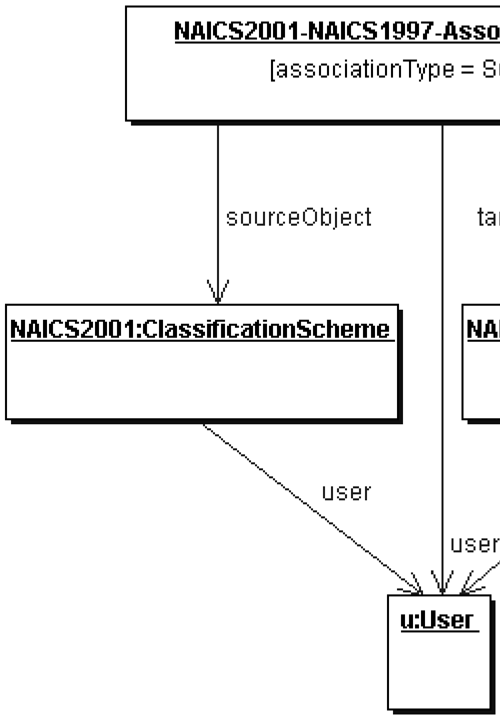
NAICS2001-NAICS1997-Association:Association
[associationType = Supersedes]
sourceObject
targetObject
NAICS2001:ClassificationScheme
NAICS1997:ClassificationScheme
user
user
user
u:User

Figure 6: Example of Intramural Association

8.5 Extramural Association

The information model also allows more sophisticated use cases. For example, a User “u1” creates an Association “a” between two RegistryObjects “o1” and “o2” where association “a” is owned by User “u1”, but RegistryObjects “o1” and “o2” are owned by User “u2” and User “u3” respectively.

In this use case an Association is defined where either or both objects that are being associated are owned by a User different from the User defining the Association. Such Associations are referred to as extramural Associations. The Association class provides a convenience method called *isExtramural* that returns *"true"* if the Association instance is an extramural Association.

Figure 7 below, extends the previous example in Figure 6 for the extramural Association case. Note that it is possible for an extramural Association to have two distinct Users rather than three distinct Users as shown in Figure 7. In such case, one of the two users owns two of the three objects involved (Association, sourceObject and targetObject).

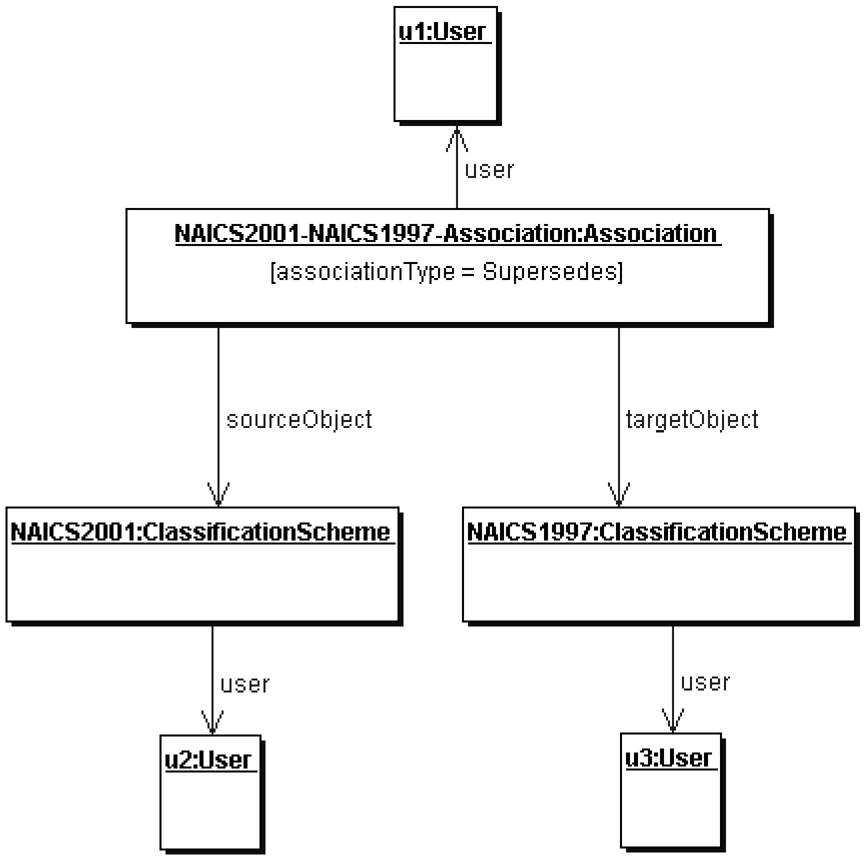
u1:User
user
NAICS2001-NAICS1997-Association:Association
[associationType = Supersedes]
sourceObject
targetObject
NAICS2001:ClassificationScheme
NAICS1997:ClassificationScheme
user
user
u2:User
u3:User

Figure 7: Example of Extramural Association

8.6 Confirmation of an Association

An Association may need to be confirmed by the parties whose objects are involved in that Association as the sourceObject or targetObject. This section describes the semantics of confirmation of an association by the parties involved.

8.6.1 Confirmation of Intramural Associations

Intramural associations may be viewed as declarations of truth and do not require any explicit steps to confirm that Association as being true. In other words, intramural associations are implicitly considered confirmed.

8.6.2 Confirmation of Extramural Associations

An extramural association may be thought of as a unilateral assertion that may not be viewed as truth until it has been confirmed by the other (extramural) parties involved (Users “u2” and “u3” in the example in section 8.5).

To confirm an extramural association, each of the extramural parties (parties that own the source or target object but do not own the Association) must submit an identical Association (clone Association) as the Association they are intending to confirm using a SubmitObjectsRequest. The clone Association must use the same id as the original Association.

8.6.3 Deleting an Extramural Associations

An Extramural Association is deleted like any other type of RegistryObject, using the RemoveObjectsRequest as defined in [ebRS]. However, in some cases deleting an extramural Association may not actually delete it but instead only revert a confirmed association to unconfirmed state.

When deleted by its owner, an extramural Association must always be deleted irrespective of its confirmation state.

When deleted by the owner of its source/target object who is not the owner of the Association itself, an extramural Association must become unconfirmed.

8.7 Visibility of Unconfirmed Associations

Unconfirmed extramural Associations are visible to third party registry clients. Third party registry clients can determine the confirmation state of an extramural Association and decide whether to trust that Association or not.

8.8 Possible Confirmation States

Assume the most general case where there are three distinct User instances as shown in Figure 7 for an extramural Association. The extramural Association needs to be confirmed by both extramural parties (Users “u2” and “u3” in example) in order to be fully confirmed. The attributes *isConfirmedBySourceOwner* and *isConfirmedByTargetOwner* in the Association class provide access to the confirmation state for both the sourceObject and targetObject. A convenience method called *isConfirmed* provides a way to determine whether the Association is fully confirmed or not. So there are the following four possibilities related to the confirmation state of an extramural Association:

- The Association is confirmed neither by the owner of the *sourceObject* nor by the owner of the *targetObject*.
- The Association is confirmed by the owner of the *sourceObject* but it is not confirmed by the owner of the *targetObject*.
- The Association is not confirmed by the owner of the *sourceObject* but it is confirmed by the owner of the *targetObject*.

- The Association is confirmed by both the owner of the *sourceObject* and the owner of the *targetObject*. This is the only state where the Association is fully confirmed.

8.9 Class Association

Super Classes:

RegistryObject

Association instances are used to define many-to-many associations among RegistryObjects in the information model.

An instance of the Association class represents an association between two RegistryObjects.

8.9.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
associationType	ObjectRef	Yes		Client	No
sourceObject	ObjectRef	Yes		Client	No
targetObject	ObjectRef	Yes		Client	No
isConfirmedBy-SourceOwner	boolean	No	false	Registry	No
isConfirmedBy-TargetOwner	boolean	No	false	Registry	No

8.9.2 Attribute associationType

Each Association must have an *associationType* attribute that identifies the type of that association. The value of the associationType attribute MUST be a reference to a ClassificationNode within the canonical AssociationType ClassificationScheme as described in appendix A.2.

8.9.2.1 Canonical Association Types

The following table lists canonical association types. These canonical association types are defined as a *ClassificationScheme* called AssociationType. While the ObjectType scheme may easily be extended, a Registry MUST support the association types as defined by the AssociationType scheme referenced in appendix A.2.

8.9.3 Attribute sourceObject

Each Association must have a *sourceObject* attribute that references the RegistryObject instance that is the source of that Association.

8.9.4 Attribute targetObject

Each Association must have a *targetObject* attribute that references the RegistryObject instance that is the target of that Association.

8.9.5 Attribute isConfirmedBySourceOwner

Each Association may have an *isConfirmedBySourceOwner* attribute that is set by the registry to be true if the Association has been confirmed by the owner of the sourceObject. For intramural Associations this attribute is always *true*. This attribute must be present when the object is retrieved from the registry. This attribute must be ignored if specified by the client when the object is submitted to the registry.

8.9.6 Attribute isConfirmedByTargetOwner

Each Association may have an *isConfirmedByTargetOwner* attribute that is set by the registry to be true if the association has been confirmed by the owner of the *targetObject*. For intramural Associations this attribute is always *true*. This attribute must be present when the object is retrieved from the registry. This attribute must be ignored if specified by the client when the object is submitted to the registry.

Method Summary of Association	
`Boolean`	**`isConfirmed`**`()` Returns true if isConfirmedBySourceOwner and isConfirmedByTargetOwner attributes are both true. For intramural Associations always returns true.
`Boolean`	**`isExtramural`**`()` Returns true if the sourceObject and/or the targetObject are owned by a User that is different from the User that created the Association.

9 Classification Information Model

This section describes how the information model supports Classification of RegistryObject.

A RegistryObject may be classified in many ways. For example the RegistryObject for the same Collaboration Protocol Profile (CPP) may be classified by its industry, by the products it sells and by its geographical location.

A general ClassificationScheme can be viewed as a Classification tree. In the example shown in Figure 8, RegistryObject instances representing Collaboration Protocol Profiles are shown as shaded boxes. Each Collaboration Protocol Profile represents an automobile manufacturer. Each Collaboration Protocol Profile is classified by the ClassificationNode named “Automotive” under the ClassificationScheme instance with name “Industry.” Furthermore, the US Automobile manufacturers are classified by the “US” ClassificationNode under the ClassificationScheme with name “Geography.” Similarly, a European automobile manufacturer is classified by the “Europe” ClassificationNode under the ClassificationScheme with name “Geography.”

The example shows how a RegistryObject may be classified by multiple ClassificationNode instances under multiple ClassificationScheme instances (e.g., Industry, Geography).

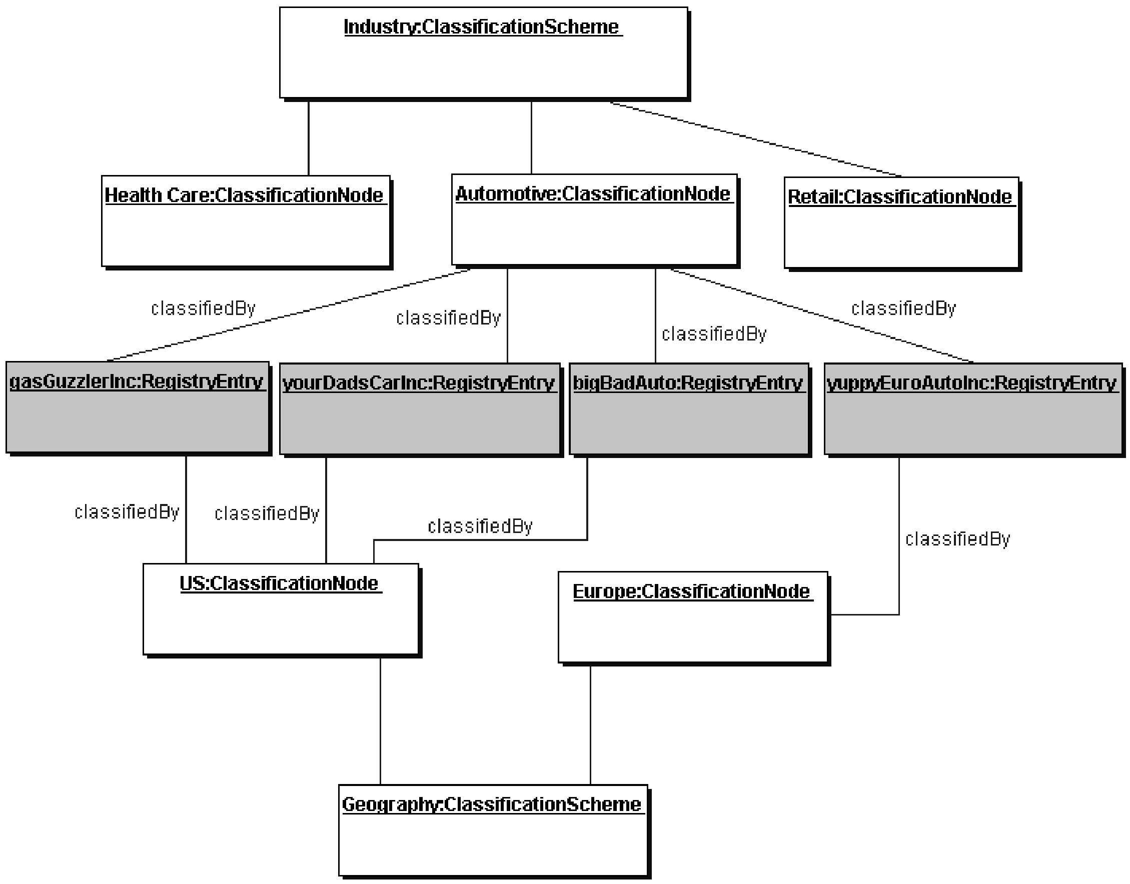
Industry:ClassificationScheme
Health Care:ClassificationNode
Automotive:ClassificationNode
Retail:ClassificationNode
classifiedBy
classifiedBy
classifiedBy
classifiedBy
gasGuzzlerInc:RegistryEntry
yourDadsCarInc:RegistryEntry
bigBadAuto:RegistryEntry
yuppyEuroAutoInc:RegistryEntry
classifiedBy
classifiedBy
classifiedBy
classifiedBy
US:ClassificationNode
Europe:ClassificationNode
Geography:ClassificationScheme

Figure 8: Example showing a *Classification* Tree

It is important to point out that the shaded nodes (gasGuzzlerInc, yourDadsCarInc etc.) are not part of the Classification tree. The leaf nodes of the Classification tree are Health Care, Automotive, Retail, US and Europe. The shaded nodes are associated with the Classification tree via a Classification Instance that is not shown in the picture.

In order to support a general ClassificationScheme that can support single level as well as multi-level Classifications, the information model defines the classes and relationships shown in Figure 9.

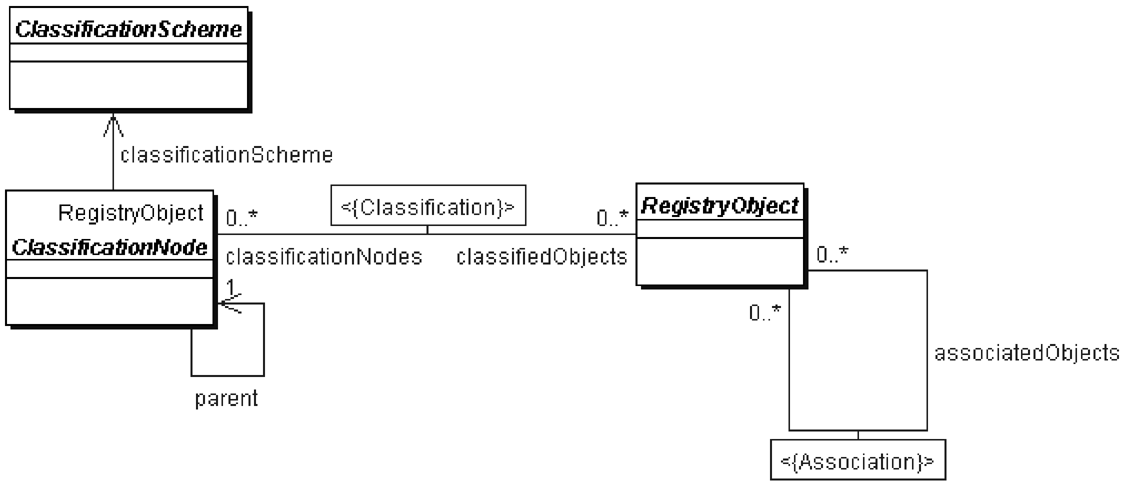

Figure 9: Information Model *Classification* View

A Classification is somewhat like a specialized form of an Association. Figure 10 shows an example of an ExtrinsicObject Instance for a Collaboration Protocol Profile (CPP) object that is classified by a ClassificationNode representing the Industry that it belongs to.

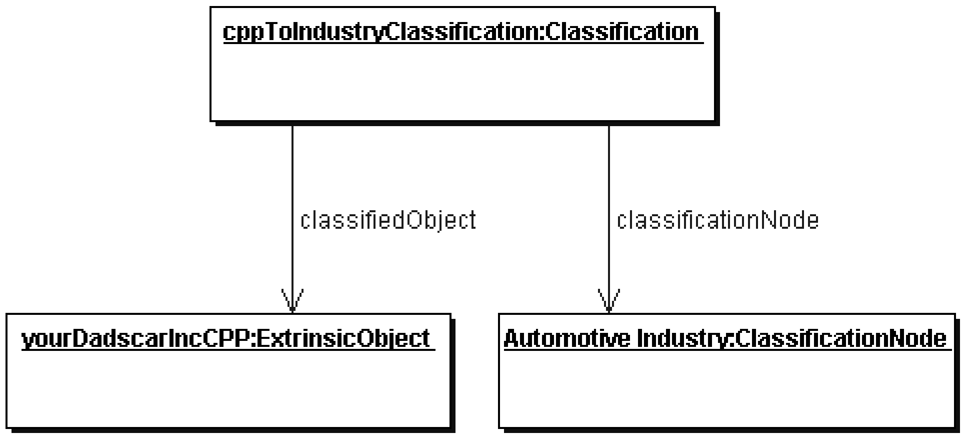
cppToIndustryClassification:Classification
classifiedObject
classificationNode
yourDadscarIncCPP:ExtrinsicObject
Automotive Industry:ClassificationNode

Figure 10: Classification *Instance* Diagram

9.1 Class ClassificationScheme

Base classes:

RegistryEntry, RegistryObject

A ClassificationScheme instance describes a registered taxonomy. The taxonomy hierarchy may be defined internally to the registry by instances of ClassificationNode or it may be defined externally to the Registry, in which case the structure and values of the taxonomy elements are not known to the Registry.

In the first case the classification scheme is defined to be *internal* and in the second case the classification scheme is defined to be *external*.

The ClassificationScheme class inherits attributes and methods from the RegistryObject and RegistryEntry classes.

9.1.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
isInternal	Boolean	Yes		Client	No
nodeType	String32	Yes		Client	No

Note that attributes inherited by ClassificationScheme class from the RegistryEntry class are not shown.

9.1.2 Attribute isInternal

When submitting a ClassificationScheme instance the Submitting Organization must declare whether the ClassificationScheme instance represents an internal or an external taxonomy. This allows the registry to validate the subsequent submissions of ClassificationNode and Classification instances in order to maintain the type of ClassificationScheme consistent throughout its lifecycle.

9.1.3 Attribute nodeType

When submitting a ClassificationScheme instance the Submitting Organization must declare the structure of taxonomy nodes within the ClassificationScheme. This attribute is an enumeration with the following values:

- *UniqueCode*. This value indicates that each node of the taxonomy has a unique code assigned to it.
- *EmbeddedPath*. This value indicates that the unique code assigned to each node of the taxonomy also encodes its path. This is the case in the NAICS taxonomy.

- *NonUniqueCode*. In some cases nodes are not unique, and it is necessary to use the full path (from ClassificationScheme to the node of interest) in order to identify the node. For example, in a geography taxonomy Moscow could be under both Russia and the USA, where there are five cities of that name in different states.

9.2 Class ClassificationNode

Base classes:

RegistryObject

ClassificationNode instances are used to define tree structures where each node in the tree is a ClassificationNode. Such Classification trees are constructed with ClassificationNode instances under a ClassificationScheme instance, and are used to define Classification schemes or ontologies.

9.2.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
parent	ObjectRef	No		Client	No
code	LongName	No		Client	No
path	String	No		Registry	No

9.2.2 Attribute parent

Each ClassificationNode may have a *parent* attribute. The parent attribute either references a parent ClassificationNode or a ClassificationScheme instance in case of first level ClassificationNode instances.

9.2.3 Attribute code

Each ClassificationNode may have a *code* attribute. The code attribute contains a code within a standard coding scheme. The code attribute of a ClassificationNode must be unique with respect to all sibling ClassificationNodes that are immediate children of the same parent ClassificationNode or ClassificationScheme.

9.2.4 Attribute path

Each ClassificationNode may have a *path* attribute. The path attribute must be present when a ClassificationNode is retrieved from the registry. The path attribute must be ignored when the path is specified by the client when the object is submitted to the registry. The path attribute contains the canonical path from the ClassificationScheme of this ClassificationNode. The path attribute of a ClassificationNode must be unique within a registry. The path syntax is defined in 9.2.6.

9.2.5 Method Summary

In addition to its attributes, the ClassificationNode class also defines the following methods.

Method Summary of ClassificationNode	
`ClassificationScheme`	**`getClassificationScheme`**`()` Get the ClassificationScheme that this ClassificationNode belongs to.
`Collection`	**`getClassifiedObjects`**`()` Get the collection of RegistryObjects classified by this ClassificationNode.
`Integer`	**`getLevelNumber()`** Gets the level number of this ClassificationNode in the classification scheme hierarchy. This method returns a positive integer and is defined for every node instance.

In Figure 8, several instances of ClassificationNode are defined (all unshaded boxes). A ClassificationNode has zero or one parent and zero or more ClassificationNodes for its immediate children. The parent of a ClassificationNode may be another ClassificationNode or a ClassificationScheme in case of first level ClassificationNodes.

9.2.6 Canonical Path Syntax

The path attribute of the ClassificationNode class contains an absolute path in a canonical representation that uniquely identifies the path leading from the ClassificationScheme to that ClassificationNode.

The canonical path representation is defined by the following BNF grammar:

```
canonicalPath ::= '/' schemeId nodePath
nodePath       ::=    '/' nodeCode
               |      '/' nodeCode ( nodePath )?
```

In the above grammar, schemeId is the id attribute of the ClassificationScheme instance, and nodeCode is defined by NCName production as defined by http://www.w3.org/TR/REC-xml-names/#NT-NCName.

9.2.6.1 Example of Canonical Path Representation

The following canonical path represents what the *path* attribute would contain for the ClassificationNode with code “United States” in the sample Geography scheme in section 9.2.6.2.

```
/Geography-id/NorthAmerica/UnitedStates
```

9.2.6.2 Sample Geography Scheme

Note that in the following examples, the *id* attributes have been chosen for ease of readability and are therefore not valid URN or UUID values.

```
<ClassificationScheme id='Geography-id' name="Geography”/>

<ClassificationNode id="NorthAmerica-id" parent="Geography-id"
code=NorthAmerica" />
<ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id"
code="UnitedStates" />

<ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
<ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
<ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
```

9.3 Class Classification

Base Classes:

RegistryObject

A Classification instance classifies a RegistryObject instance by referencing a node defined within a particular ClassificationScheme. An internal Classification will always reference the node directly, by its id, while an external Classification will reference the node indirectly by specifying a representation of its value that is unique within the external classification scheme.

The attributes and methods for the Classification class are intended to allow for representation of both internal and external classifications in order to minimize the need for a submission or a query to distinguish between internal and external classifications.

In Figure 8, Classification instances are not explicitly shown but are implied as associations between the RegistryObject instances (shaded leaf node) and the associated ClassificationNode.

9.3.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
classificationScheme	ObjectRef	for external classifications	null	Client	No
classificationNode	ObjectRef	for internal classifications	null	Client	No
classifiedObject	ObjectRef	Yes		Client	No
nodeRepresentation	LongName	for external classifications	null	Client	No

Note that attributes inherited from the base classes of this class are not shown.

9.3.2 Attribute classificationScheme

If the Classification instance represents an external classification, then the *classificationScheme* attribute is required. The classificationScheme value must reference a ClassificationScheme instance.

9.3.3 Attribute classificationNode

If the Classification instance represents an internal classification, then the *classificationNode* attribute is required. The *classificationNode* value must reference a ClassificationNode instance.

9.3.4 Attribute classifiedObject

For both internal and external classifications, the *classifiedObject* attribute is required and it references the RegistryObject instance that is classified by this Classification.

9.3.5 Attribute nodeRepresentation

If the Classification instance represents an external classification, then the *nodeRepresentation* attribute is required. It is a representation of a taxonomy element from a classification scheme. It is the responsibility of the registry to distinguish between different types of *nodeRepresentation*, like between the classification scheme node code and the classification scheme node canonical path. This allows the client to transparently use different syntaxes for *nodeRepresentation*.

9.3.6 Method Summary

In addition to its attributes, the Classification class also defines the following methods:

Fix indentation in first para of method descriptions inside tables of methods??

Return Type	Method
String	**getPath**()

	For an external classification returns a string that conforms to the canonical path syntax as specified in 9.2.6. For an internal classification, returns the value contained in the path attribute of the ClassificationNode instance identified by the classificationNode attribute.
`LongName`	**getCode**() For an external classification, returns a string that represents the declared value of the taxonomy element. It will not necessarily uniquely identify that node. For an internal classification, returns the value of the code attribute of the ClassificationNode instance identified by the classificationNode attribute.

9.3.7 Context Sensitive *Classification*

Consider the case depicted in Figure 11 where a Collaboration Protocol Profile for ACME Inc. is classified by the “Japan” ClassificationNode under the “Geography” Classification scheme. In the absence of the context for this Classification its meaning is ambiguous. Does it mean that ACME is located in Japan, or does it mean that ACME ships products to Japan, or does it have some other meaning? To address this ambiguity a Classification may optionally be associated with another ClassificationNode (in this example named isLocatedIn) that provides the missing context for the Classification. Another Collaboration Protocol Profile for MyParcelService may be classified by the “Japan” ClassificationNode where this Classification is associated with a different ClassificationNode (e.g., named shipsTo) to indicate a different context than the one used by ACME Inc.

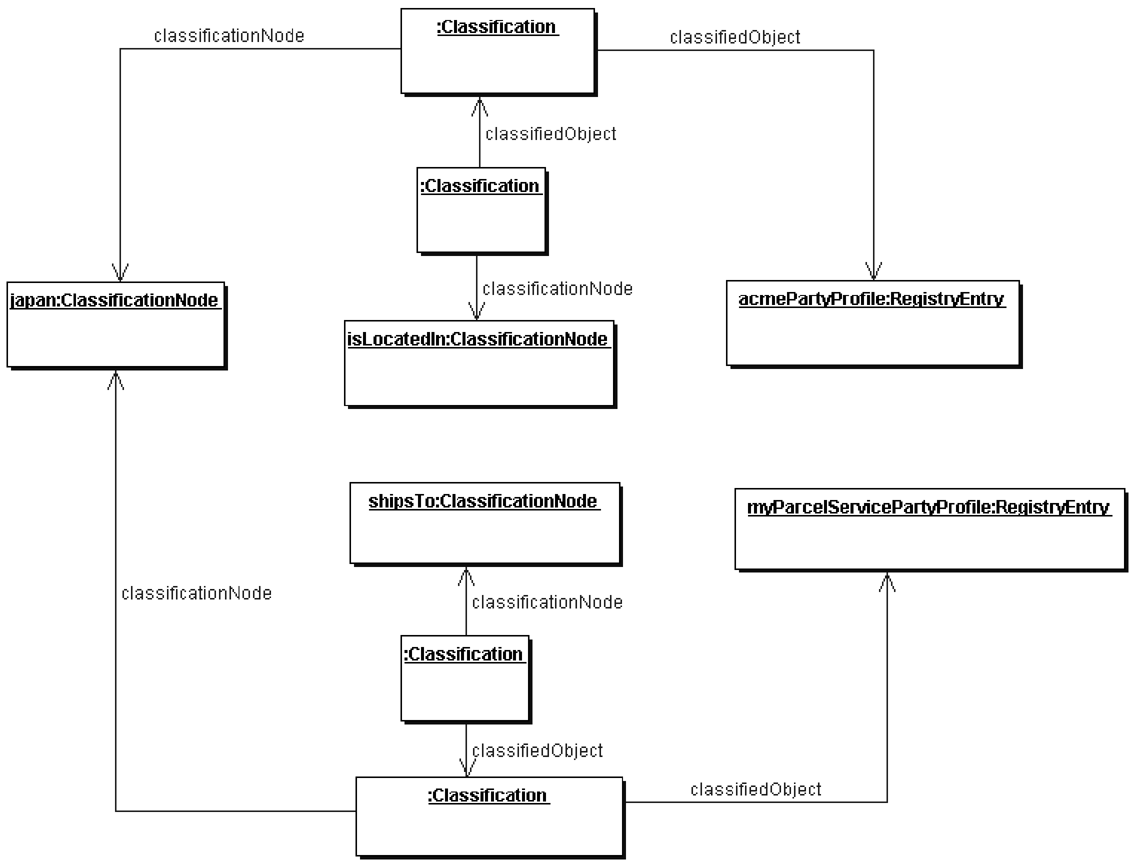

Figure 11: Context Sensitive *Classification*

Thus, in order to support the possibility of Classification within multiple contexts, a Classification is itself classified by any number of Classifications that bind the first Classification to ClassificationNodes that provide the missing contexts.

In summary, the generalized support for *Classification* schemes in the information model allows:

- o A RegistryObject to be classified by defining an internal Classification that associates it with a ClassificationNode in a ClassificationScheme.
- o A RegistryObject to be classified by defining an external Classification that associates it with a value in an external ClassificationScheme.
- o A RegistryObject to be classified along multiple facets by having multiple Classifications that associate it with multiple ClassificationNodes or value within a ClassificationScheme.
- o A Classification defined for a RegistryObject to be qualified by the contexts in which it is being classified.

9.4 Example of Classification Schemes

The following table lists some examples of possible ClassificationSchemes enabled by the information model. These schemes are based on a subset of contextual concepts identified by the ebXML Business Process and Core Components Project Teams. This list is meant to be illustrative not prescriptive.

Classification Scheme	**Usage Example**	**Standard Classification Schemes**
Industry	Find all Parties in Automotive industry	NAICS
Process	Find a ServiceInterface that implements a Process	
Product / Services	Find a Business that sells a product or offers a service	UNSPSC
Locale	Find a Supplier located in Japan	ISO 3166
Temporal	Find Supplier that can ship with 24 hours	
Role	Find All Suppliers that have a Role of “Seller”	

Table 1: Sample Classification Schemes

10 Service Information Model

This chapter describes the classes in the information model that support the registration of services. The service registration information model is flexible and supports the registration of web services as well as other types of services.

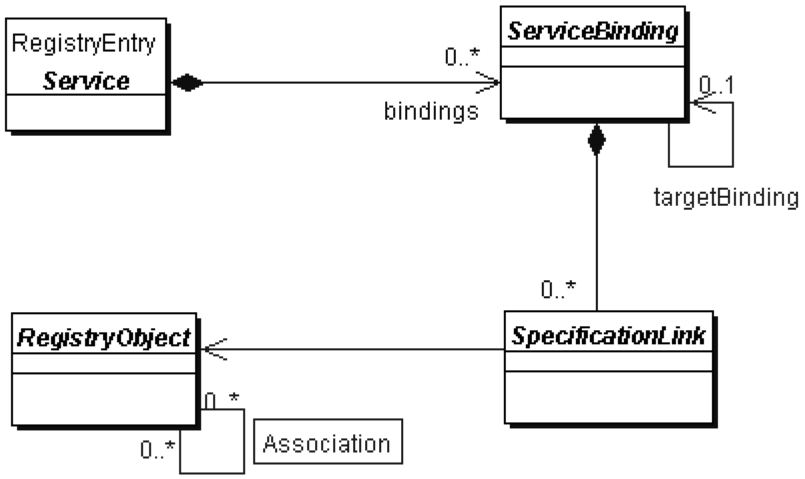

Figure 12: Service Information Model

10.1 Class Service

Super Classes:

RegistryEntry, RegistryObject

Service instances provide information on services, such as web services.

10.1.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
serviceBindings	Collection of ServiceBinding	Yes		Client	Yes

10.1.2 Attribute serviceBindings

A Service must have a *serviceBindings* attribute that defines the service bindings that provide access to that Service. Each ServiceBinding instance represents technical information on a specific way to access a specific interface offered by a Service instance.

10.2 Class ServiceBinding

Super Classes:

RegistryObject

ServiceBinding instances are RegistryObjects that represent technical information on a specific way to access a specific interface offered by a Service instance. A Service has a Collection of ServiceBindings.

The *description* attribute of ServiceBinding provides details about the relationship between several specification links comprising the Service Binding.

10.2.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
accessURI	URI	No		Client	Yes
specificationLinks	Collection of SpecificationLink	Yes		Client	Yes
targetBinding	ObjectRef	No		Client	Yes

10.2.2 Attribute accessURI

A ServiceBinding may have an *accessURI* attribute that defines the URI to access that ServiceBinding. This attribute is ignored if a targetBinding attribute is specified for the ServiceBinding. If the URI is a URL then a registry must validate the URL to be resolvable at the time of submission before accepting a ServiceBinding submission to the registry.

10.2.3 Attribute specificationLinks

A ServiceBinding must have a *specificationLinks* attribute defined that is a collection of references to SpecificationLink instances. Each SpecificationLink instance links the ServiceBinding to a particular technical specification that may be used to access the Service for the ServiceBinding.

10.2.4 Attribute targetBinding

A ServiceBinding may have a *targetBinding* attribute defined that references another ServiceBinding. A targetBinding may be specified when a service is being redirected to another service. This allows the rehosting of a service by another service provider.

10.3 Class SpecificationLink

Super Classes:

RegistryObject

A SpecificationLink provides the linkage between a ServiceBinding and one of its technical specifications that describes how to use the service using the ServiceBinding. For example, a ServiceBinding may have SpecificationLink instances that describe how to access the service using a technical specification such as a WSDL document or a CORBA IDL document.

10.3.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
specificationObject	ObjectRef	Yes		Client	Yes
usageDescription	InternationalString	No		Client	Yes
usageParameters	Collection of FreeFormText	No		Client	Yes

10.3.2 Attribute specificationObject

A SpecificationLink instance must have a *specificationObject* attribute that provides a reference to a RegistryObject instance that provides a technical specification for the parent ServiceBinding. Typically, this is an ExtrinsicObject instance representing the technical specification (e.g., a WSDL document). It may also be an ExternalLink object in case the technical specification is a resource that is external to the registry.

10.3.3 Attribute usageDescription

A SpecificationLink instance may have a *usageDescription* attribute that provides a textual description of how to use the optional usageParameters attribute described next. The usageDescription is of type InternationalString, thus allowing the description to be in multiple languages.

10.3.4 Attribute usageParameters

A SpecificationLink instance may have a *usageParameters* attribute that provides a collection of Strings representing the instance specific parameters needed to use the technical specification (e.g., a WSDL document) specified by this SpecificationLink object.

11 Event Information Model

This chapter defines the information model classes that support the registry Event Notification feature. These classes include AuditableEvent, Subscription, Selector and Action. They constitute the foundation of the Event Notification information model.

Figure 13 shows how a Subscription may be defined that uses a pre-configured AdhocQuery instance as a selector to select the AuditableEvents of interest to the subscriber and an Action to deliver the selected events to the subscriber. The Action may deliver the events by using its endPoint attribute to invoke a registered ServiceBinding to a registered Service or by sending the vents to an email address.

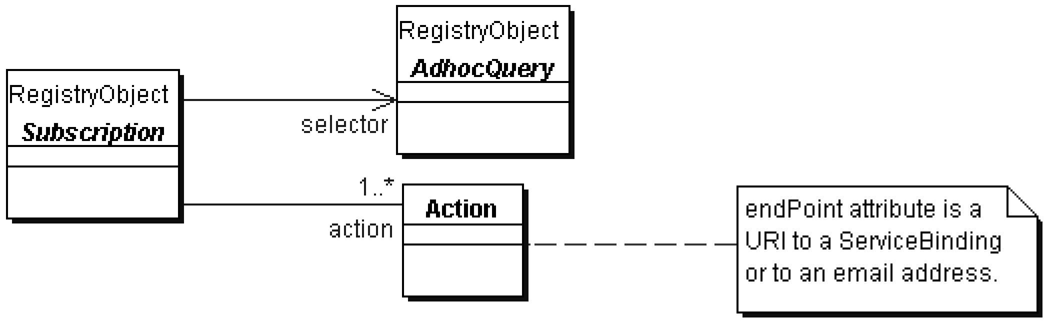

Figure 13: Event Information Model

11.1 Class AuditableEvent

Super Classes:

RegistryObject

AuditableEvent instances provide a long-term record of events that effected a change in a RegistryObject. A RegistryObject is associated with an ordered Collection of AuditableEvent instances that provide a complete audit trail for that RegistryObject.

AuditableEvents are usually a result of a client-initiated request. AuditableEvent instances are generated by the Registry Service to log such Events.

Often such events effect a change in the life cycle of a RegistryObject. For example a client request could Create, Update, Deprecate or Delete a RegistryObject. An AuditableEvent is created if and only if a request creates or alters the content or ownership of a RegistryObject. Read-only requests do not generate an AuditableEvent.

11.1.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
eventType	LongName	Yes		Registry	No
affectedObjects	Collection of	Yes		Registry	No

	ObjectRef				
requestId	URI	Yes		Registry	No
timestamp	dateTime	Yes		Registry	No
user	ObjectRef	Yes		Registry	No

11.1.2 Attribute eventType

Each AuditableEvent must have an eventType attribute which identifies the type of event recorded by the AuditableEvent.

11.1.2.1 Pre-defined Auditable Event Types

The following table lists pre-defined auditable event types. A *Registry* MUST support the event types listed below.

Name	Description
Approved	An Event that marks the approval of a RegistryObject.
Created	An Event that marks the creation of a RegistryObject.
Deleted	An Event that marks the deletion of a RegistryObject.
Deprecated	An Event that marks the deprecation of a RegistryObject.
Downloaded	An Event that marks the downloading of a RegistryObject.
Relocated	An Event that marks the relocation of a RegistryObject.
Updated	An Event that that marks the updating of a RegistryObject.
Versioned	An Event that marks the versioning of a RegistryObject.

11.1.3 Attribute affectedObjects

Each AuditableEvent must have an *affectedObjects* attribute that identifies the collection of RegistryObjects instances that were affected by this event.

11.1.4 Attribute requestId

Each AuditableEvent must have a *requestId* attribute that identifies the client request instance that affected this event.

11.1.5 Attribute timestamp

Each AuditableEvent must have a *timestamp* attribute that records the date and time that this event occurred.

11.1.6 Attribute user

Each AuditableEvent must have a *user* attribute that identifies the User that sent the request that generated this event affecting the RegistryObject instance.

11.2 Class Subscription

Super Classes:

RegistryObject

Subscription instances are RegistryObjects that define a User's interest in certain types of AuditableEvents. A User may create a subscription with a registry if she wishes to receive notification for a specific type of event.

11.2.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
actions	Collection of Action	Yes, may be empty		Client	Yes
endDate	dateTime	No		Client	Yes
notificationInterval	duration	No	P1D (1 day)	Client	No
selector	AdhocQuery	Yes		Client	No
startDate	dateTime	No	Current time	Client	Yes

11.2.2 Attribute action

A Subscription instance must have an *actions* attribute that is a Collection of zero or more Action instances. An Action instance describes what action the registry must take when an event matching the Subscription transpires. The Action class is described in section 11.4.

11.2.3 Attribute endDate

This attribute denotes the time after which the subscription expires and is no longer active. If this attribute is missing the subscription never expires.

11.2.4 Attribute notificationInterval

This attribute denotes the duration that a registry must wait between delivering successive notifications to the client. The client specifies this attribute in order to control the frequency of notification communication between registry and client. If this attribute is missing, sending of notifications should happen as soon as relevant events occur.

11.2.5 Attribute selector

This attribute defines the selection criterea that determines which events match this Subscription and are of interest to the User. The selector attribute references a pre-defined query that is stored in the registry as an instance of the AdhocQuery class. This AdhocQuery instance specifies or "selects" events that are of interest to the subscriber. The AdhocQueryClass is described in section **Error! Reference source not found.**.

11.2.6 Attribute startDate

This attribute denotes the time at which the subscription becomes active. If this attribute is missing subscription starts immediately.

11.3 Class AdhocQuery

This abstract class is a sub-class of RegistryObject class and is the base class for all AdhocQuery classes supported by the registry. Instances of this class represent an ad hoc query that may be used for discover RegistryObjects within the registry. Instances of AdhocQuery may be stored in registry like other RegistryObjects. Such stored AdhocQuery instances are similar in purpose to the concept of stored procedures in relational databases.

11.3.1 Method Summary

In addition to its attributes, the AdhocQUery class also defines the following methods.

Method Summary for RegistryObject	
`String`	**getQuery**`()` Gets the query String for this AdhocQuery instances. This may be an SQL or Filter query string as described by [ebRS].

11.4 Class Action

The Action class is an abstract base class that specifies what the registry must do when an event matching the action's Subscription transpires. A registry uses Actions within a Subscription to asynchronously deliver event Notifications to the subscriber.

If no Actions are defined within the Subscription that implies that the user does not wish to be notified asynchronously by the registry and instead intends to periodically poll the registry and pull the pending Notifications.

This class does not currently define any attributes or method.

11.5 Class NotifyAction

The NotifyAction class is a sub-class of Action class. An instance of NotifyAction represents an Action that the registry must perform in order to notify the subscriber of a Subscription of the events of interest to that subscriber.

11.5.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
endPoint	URI	YES		Client	
notificationOption	Enumeration	No	“ObjectRefs”	Client	Yes

11.5.2 Attribute endPoint

This attribute specifies a URI that identifies an service end point that may be used by the registry to deliver notifications. Currently this attribute can either be a “mailto” URI (e.g. mailto:someone@acme.com) or an “urn:uuid” URI. If it is a “mailto” URI then the registry must use the specified email address to deliver the notification via email. If it is a “urn:uuid” URI then it must be a reference to a ServiceBinding object to a Service that implements the RegistryClient interface as defined by [ebRS]. In this case the registry must deliver the notification byinvoking the onResponse method of the RegistryClient interface.

11.5.3 Attribute notificationOption

This attribute controls the specific type of event notification content desired by the subscriber. It is used by the subscriber to control the granularity of event notification content communicated by the registry to the subscriber.

11.5.3.1 Pre-defined notificationOption Values

The following table lists pre-defined notificationOption values.

Name	Description
ObjectRefs	Indicates that the subscriber wants to receive only references to RegistryObjects that match the Subscription within a notification.
Objects	Indicates that the subscriber wants to receive actual

	RegistryObjects that match the Subscription within a notification.

12 Cooperating Registries Information Model

This chapter describes the classes in the information model that support the cooperating registries capability.

12.1.1 Class Registry

Super Classes:

RegistryEntry

Registry instances are used to represent a single physical OASIS ebXML registry.

12.1.1.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
catalogingLatency	duration	No	P1D (1 day)	Registry	Yes
eventNotification Supported	boolean	No	false	Registry	Yes
objectRelocation Supported	boolean	No	false	Registry	Yes
objectReplication Supported	boolean	No	false	Registry	Yes
operator	ObjectRef	Yes		Registry	Yes
replicationSync Latency	duration	No	P1D (1 day)	Registry	Yes
specification Version	Sring8	Yes		Registry	Yes
sqlQuerySupported	boolean	No	false	Registry	Yes

12.1.1.2 Attribute catalogingLatency

Each Registry instance may have an attribute named *catalogingLatency* that specifies the maximum latency between the time a submission is made to the registry and the time it gets cataloged by any cataloging services defined for the objects within the submission.

12.1.1.3 Attribute eventNotificationSupported

Each Registry instance may have an attribute named *eventNotificationSupported* that decalres whether the registry supports the optional Event Notification feature.

12.1.1.4 Attribute objectRelocationSupported

Each Registry instance may have an attribute named *objectRelocationSupported* that decalres whether the registry supports the optional Object Relocation feature.

12.1.1.5 Attribute objectReplicationSupported

Each Registry instance may have an attribute named *objectReplicationSupported* that decalres whether the registry supports the optional Object Replication feature.

12.1.1.6 Attribute operator

Each Registry instance must have an attribute named *operator* that is a reference to the Organization instance representing the organization for the registry's operator. Since the same Organization may operate multiple registries, it is possible that the home registry for the Organization referenced by operator may not be the local registry.

12.1.1.7 Attribute replicationSyncLatency

Each Registry instance may have an attribute named *replicationSyncLatency* that specifies the maximum latency between the time when an original object changes and the time when its replica object within the registry gets updated to synchronize with the new state of the original object.

12.1.1.8 Attribute specificationVersion

Each Registry instance must have an attribute named *specificationVersion* that is the version of the ebXML Registry Services Specification [ebRS].

12.1.1.9 Attribute sqlQuerySupported

Each Registry instance may have an attribute named *sqlQuerySupported* that decalres whether the registry supports the optional SQL Query feature.

12.1.2 Class Federation

Super Classes:

RegistryEntry

Federation instances are used to represent a registry federation.

12.1.2.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
replicationSync Latency	duration	No	P1D (1 day)	Client	Yes

12.1.2.2 Attribute replicationSyncLatency

Each Federation instance may specify a *replicationSyncLatency* attribute that describes the time duration that is the amount of time within which a member of this Federation must synchronize itself with the current state of the Federation. Members of the Federation may use this parameter to periodically synchronize the federation metadata they must cache locally about the state of the Federation and its members. Such synchronization may be based upon the registry event notification capability.

12.1.3 Federation Configuration

A federation is created by the creation of a Federation instance. Membership of a registry within a federation is established by creating an Association between the Registry instances for the registry seeking membership with the Federation instance. The Association must have its associationType be "HasFederationMember", the federation instance as its sourceObject and the Registry instance as its targetObject as shown in Figure 14.

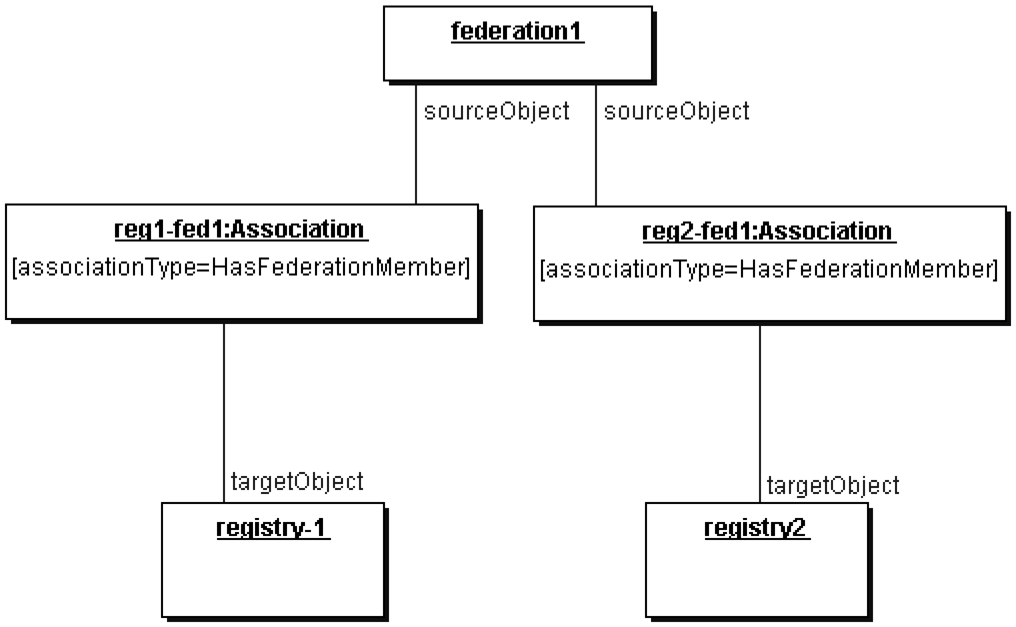

Figure 14: Federation Information Model

13 Access Control Information Model

This chapter defines the Access Control Information Model used by the registry to control access to RegistryObjects and RepositoryItems managed by it.

This specification first defines an abstract Access Control Model that enables access control policies to be defined and associated with RegistryObjects.

Next, it defines a normative and required binding of that abstract model to [XACML].

Finally, it defines how a registry may support additional bindings to custom access control technologies.

13.1 Terminology

The Access Control Model attempts to reuse terms defined by [XACML] wherever possible. The definition of some key terms are duplicated here from [XACML] for convenience of the reader:

Access - Performing an ***action***. An example is a user performing a *delete action* on a RegistryObject.

Access control - Controlling ***access*** in accordance with a ***policy***. An example is preventing a user from performing a *delete action* on a RegistryObject that is not owned by that user.

Action - An operation on a ***resource***. An example is the *delete action* on a RegistryObject.

Attribute - Characteristic of a ***subject***, ***resource, action***. Some examples are:

- *id attribute* of a subject.
- *role attribute* of a subject.
- *group attribute* of a subject.
- *id attribute* of a RegistryObject resource.

Policy - A set of ***rules.*** May be a component of a ***policy set***

Policy set - A set of ***policies,*** other ***policy sets.*** May be a component of another ***policy set***

Resource - Data, service or system component. Examples are:

- A *RegistryObject resource*
- A *RepositoryItem resource*

Subject - An actor whose ***attributes*** may be referenced by within a Policy definition. Examples are:

- A User instance within the registry

13.2 Resources

A registry must control access to the following types of resources:

- *RegistryObject resource* is any instance of RegistryObject class or its sub-classes. Each RegistryObject resource references an Access Control Policy that controls all access to that object.
- *RegistryEntry resource* is any instance of RepositoryItem class. By default, access control to a RepositoryItem is managed by the same Access Control Policy as its ExtrinsicObject.

A Registry must support each and every attribute of the RegistryObject class and all of its sub-classes within its Access Control Policies. In addition a registry must support the following additional resource attributes.

13.2.1 Attribute owner

The *owner* attribute of a Resource carries the value of id attribute of the User instance within the registry that represents the owner of the resource.

13.2.2 Attribute selector

The *selector* attribute of a Resource carries a string representing a query as define by a sub-type of AdhocQueryType in [ebRS].

13.3 Actions

A registry must support the following actions as operations on RegistryObject and RepositoryItem resources managed by the registry.

13.3.1 Create Action

The *create action* creates a RegistryObject or a RepositoryItem. A submitObjects operation performed on the LifeCycleManager interface of the registry result in a *create action.*

13.3.2 Read Action

The *read action* reads a RegistryObject or a RepositoryItem without having any impact on its state. An operation performed on the QueryManager interface of the registry result in a *read action.*

13.3.3 Update Action

The *update action* updates or modifies the state of a RegistryObject or a RepositoryItem. An updateObjects operation performed on the LifeCycleManager interface of the registry result in a *update action.*

13.3.4 Delete Action

The *delete action* deletes a RegistryObject or a RepositoryItem. A removeObjects operation performed on the LifeCycleManager interface of the registry result in a *delete action.*

13.3.5 Approve Action

The *approve action* approves a RegistryObject. An approveObjects operation performed on the LifeCycleManager interface of the registry result in a *approve action.*

13.3.6 Reference Action

The *reference action* creates a reference to a RegistryObject. A submitObjects or updateObjects operation performed on the LifeCycleManager interface of the registry may result in a *reference action.* An example of a reference action is when an Association is created that references a RegistryObject resource as its source or target object.

13.3.7 Deprecate Action

The *deprecate action* deprecates a RegistryObject. A deprecateObjects operation performed on the LifeCycleManager interface of the registry result in a *deprecate action.*

13.3.8 Undeprecate Action

The *undeprecate action* undeprecates a previously deprecated RegistryObject. An undeprecateObjects operation performed on the LifeCycleManager interface of the registry result in a *undeprecate action.*

13.4 Subjects

A registry must support the following Subject attributes within its Access Control Policies. In addition a registry may support additional subject attributes.

13.4.1 Attribute id

The *identity* attribute of a Subject carries the value of id attribute of a User instance within the registry.

13.4.2 Attribute group

The *group* attribute of a Subject carries the value of the code attribute of a ClassificationNode within the canonical SubjectGroup ClassificationScheme (see appendix A.9) within the registry.

13.4.2.1 Assigning To Users to Groups

Arbitrary groups may be defined by extending the canonical SubjectGroup ClassificationScheme. Groups may be assigned to registered users by classifying their User instance with a ClassificationNode within the canonical SubjectGroup ClassificationScheme.

13.4.3 Attribute role

The *role* attribute of a Subject carries the value of the code attribute of a ClassificationNode within the canonical SubjectRole ClassificationScheme (see appendix A.8) within the registry.

13.4.3.1 Assigning Roles To Users

Arbitrary roles may be defined by extending the canonical SubjectRole ClassificationScheme. Roles may be assigned to registered users by classifying their User instance with a ClassificationNode within the canonical SubjectRole ClassificationScheme.

13.5 Use Cases for Access Control Policies

The following are some comon use cases for access control policy:

13.5.1 Default Access Control Policy

Define a default access control policy that gives *read access* to any one and access to all actions to ContentOwner and Registry Administrator. This access control policy implicitly applies to any resource that does not explicitly have a custom Access Control Policy defined for it.

13.5.2 Restrict Read Access To Specified Subjects

Define a custom access control policy to restrict *read access* to a resource to specified user(s), group(s) and/or role(s).

13.5.3 Grant Update and/or Delete Access To Specified Subjects

Define a custom access control policy to grant *update* and/or *delete access* to a resource to specified user(s), group(s) and/or role(s).

13.6 Abstract Access Control Model

Every RegistryObject is associated with exactly one Access Control Policy that governs "who" is authorized to perform "what" action on that RegistryObject. The abstract Access Control Model allows the Access Control Policy to be defined in any arbitrary format as long as it is represented in the registry as a repositoryItem and its corresponding ExtrinsicObject. The objectType attribute of this ExtrinsicObject must reference the AccessControlPolicy node in the canonical ObjectType ClassificationScheme or one of its descendents. This distinguishes Access Control Policy objects from other ExtrinsicObject instances.

13.7 Access Control Policy for a RegistryObject

A RegistryObject may be associated with an Access Control Policy by a special Association with the canonical associationType of "AccessControlPolicyFor". This association has the reference to the ExtrinsicObject representing the Access Control Policy as the value of its sourceObject and has the reference to the RegistryObject as the value of its targetObject attribute.

If aa RegistryObject does not have an Access Control Policy explicitly associated with it, then it is implicitly associated with the default Access Control Policy defined for the registry.

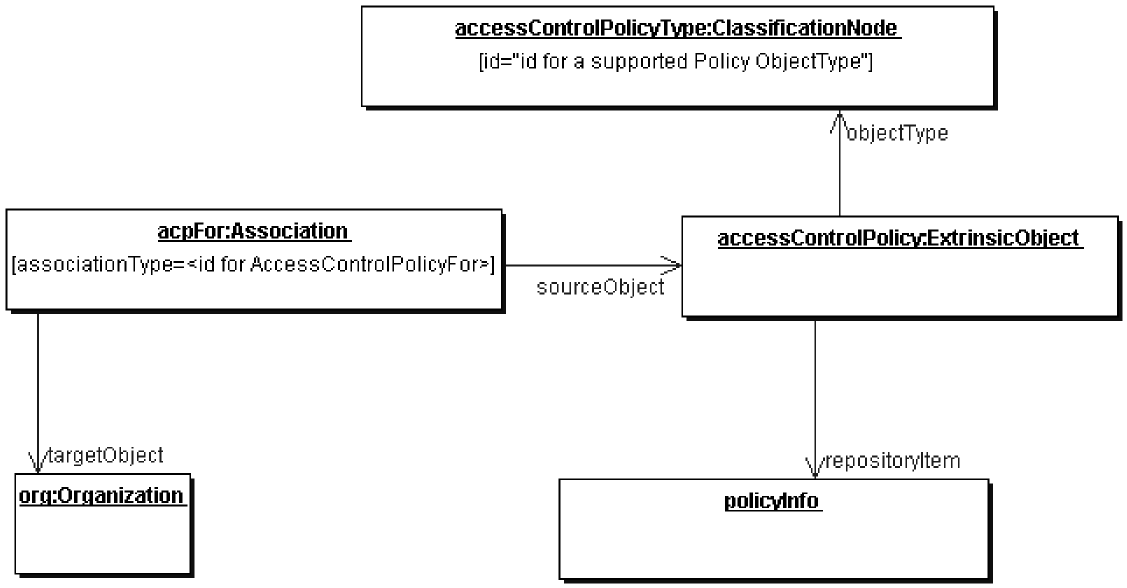
accessControlPolicyType:ClassificationNode
[id="id for a supported Policy ObjectType"]
objectType
acpFor:Association
[associationType=<id for AccessControlPolicyFor>]
sourceObject
accessControlPolicy:ExtrinsicObject
targetObject
repositoryItem
org:Organization
policyInfo

Figure 15: Instance Diagram for Abstract Access Control Information Model

Figure 15 shows an instance diagram where an Organization instance *org* references an ExtrinsicObject instance *accessControlPolicy* as its Access Control Policy object. The *accessControlPolicy* object has its objectType attribute referencing a node in the canonical ObjectType ClassificationScheme that represents a supported Access Control Policy format. The *accessControlPolicy* ExtrinsicObject has a repositoryItem defining its access control policy information in a specific format.

13.7.1 Access Control Policy for a RepositoryItem

By default, access control to a RepositoryItem is managed by the Access Control Policy associated with its ExtrinsicObject that provides metadata for the RepositoryItem. A RepositoryItem may have an Access Control Policy separate from its ExtrinsicObject. In such case, the Access Control Policy for the RespoistoryItem is referenced via a Special Slot on its ExtrinsicObject. This special Slot has "repositoryItemACP" as its name and the id of the ExtrinsicObject representing the Access Control Policy for the RepositoryItem as its value.

13.7.2 Default Access Control Policy

A registry must support the default Access Control Policy.

The default access control policy applies to any RegistryObject that does not excplicitly reference a specific access control policy via its accessControlPOlicy attribute. This is the case when a RegistryObject has a null value for its accessControlPolicy attribute.

The following list summarizes the default AccessControlPolicy semantic that a registry should implement:

- Only a Registered User is granted access to create actions.
- An unauthenticated Registry Client is granted access to read actions. The Registry must assign the default GuestReader role to such Registry Clients.
- A Registered User has access to all actions on Registry Objects submitted by the Registered User. Such Registered Users have the role of ContentOwner for the RegistryObject.
- The RegistryAdministrator and Registry Authority have access to all actions on all Registry Objects.

A registry may have a default access control policy that differs from the above semantics.

13.7.3 Root Access Control Policy

A registry must have a root Access Control Policy that bootstraps the Access Control Model by controlling access to Access Control Policies.

As described in Figure 15, an access control policy is an ExtrinsicObject which contains a pointer to a repository item. The access control policies themselves are created, updated, and deleted.

To define who may create access control policies pertaining to specified resources, it is necessary to have one or more administrative Access Control Policies. Such policies restrict Registry Users from creating access control policies to unauthorized resources. This version of the Registry specifications defines a single Root Access Control Policy that allows all actions on Access Control Policies for a resource under the following conditions:

- Subject has a role of ContentOwner for the resource
- Subject has a role of RegistryAdministrator

13.8 Access Control Model: [XACML] Binding

A registry may support custom access control policies based upon a normative though optional binding of the Access Control Model to [XACML].

This section defines the normative though optional binding of the abstract Access Control Model to [XACML]. This section assumes the reader is familiar with [XACML].

This binding to [XACML] enables a flexible access control mechanism that supports access control policy definition from the simples to the most sophisticated use cases.

In this binding the policyInfo repositoryItem in the abstract Access Control Model must be one of the following:

- A PolicySet as defined by [XACML]
- A Policy as defined by [XACML]

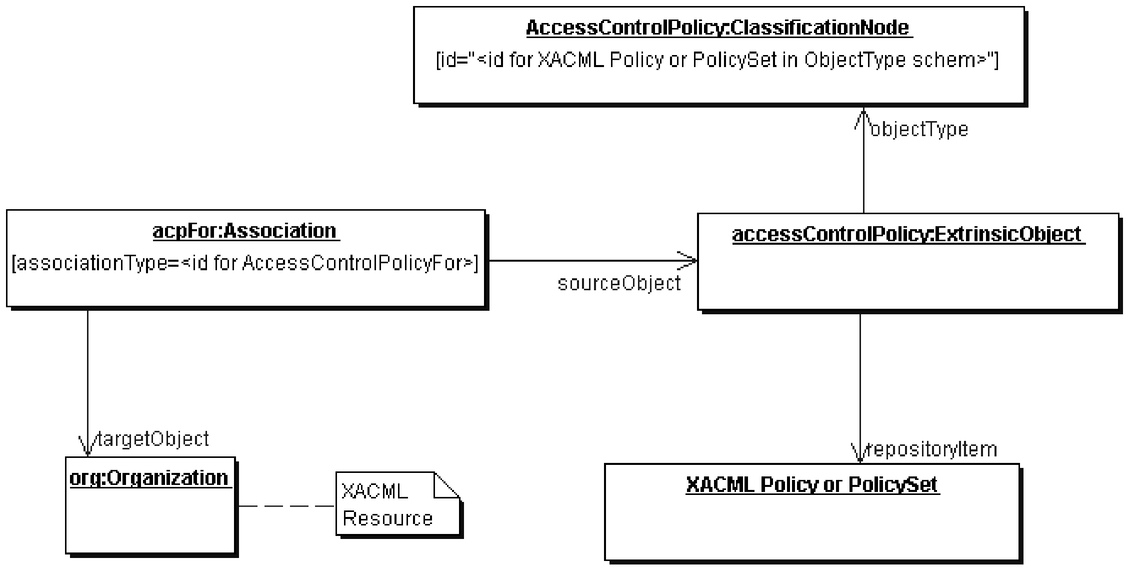
AccessControlPolicy:ClassificationNode
[id="<id for XACML Policy or PolicySet in ObjectType schem>"]
objectType
acpFor:Association
[associationType=<id for AccessControlPolicyFor>]
sourceObject
accessControlPolicy:ExtrinsicObject
targetObject
repositoryItem
org:Organization
XACML Resource
XACML Policy or PolicySet

Figure 16: Access Control Information Model: [XACML] Binding

13.8.1 Resource Binding

[XACML] defines an element called ResourceAttributeDesignator that identifies the type of resource attribute being specified in a ResourceMatch or Apply element.

The resource attributes defined by the abstract Access Control Model map to the following ResourceAttributeDesignator definitions:

Resource Attribute	ResourceAttributeDesignator	DataType
owner	urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:resource:owner	`http://www.w3.org/2001/XMLSchema#anyURI`
selector	urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:resource:selector	`http://www.w3.org/2001/XMLSchema#string`
<attribute>	urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:resource:<attribute>	`As defined by attribute definition`

Table 2: Resource Binding to [XACML]

The resource attribute <attribute> in last row in the table represents any attribute defined by the RegistryObject type or one of its sub-types.

13.8.2 Action Binding

[XACML] defines an element called ActionAttributeDesignator that identifies the type of action being specified within in an ActionMatch or Apply element.

The actions defined by the abstract Access Control Model map to the following AttributeId and AttributeValue in the ActionMatch definitions:

Registry Action	ActionMatch.ActionAttributeDesignator.AttributeId	AttributeValue
Create	`urn:oasis:names:tc:xacml:1.0:action:action-id`	`create`
Read	`urn:oasis:names:tc:xacml:1.0:action:action-id`	`read`
Update	`urn:oasis:names:tc:xacml:1.0:action:action-id`	`update`
Delete	`urn:oasis:names:tc:xacml:1.0:action:action-id`	`delete`
Approve	`urn:oasis:names:tc:xacml:1.0:action:action-id`	`approve`
Deprecate	`urn:oasis:names:tc:xacml:1.0:action:action-id`	`deprecate`
Undeprecate	`urn:oasis:names:tc:xacml:1.0:action:action-id`	`undeprecate`

Table 3: Action Binding to [XACML]

13.8.3 Subject Binding

[XACML] defines an element called SubjectAttributeDesignator that identifies the type of subject attribute being specified in a SubjectMatch or Apply element.

The subjects defined by the abstract Access Control Model map to the following SubjectAttributeDesignator definitions:

Subject Attribute	SubjectAttributeDesignator	DataType
id	urn:oasis:names:tc:xacml:1.0:subject:subject-id	http://www.w3.org/2001/XMLSchema#anyURI
role	urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:subject:role	http://www.w3.org/2001/XMLSchema#string
group	urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:subject:group	http://www.w3.org/2001/XMLSchema#string

Table 4: Subject Binding to [XACML]

13.8.4 Constraints on [XACML] Binding

This specification normatively defines the following constraints on the binding of the Access Control Model to [XACML]. These constraints may be relaxed in future versions of this specification.

- All Policy and PolicySet definitions must reside with an ebXML registry as RepositoryItems.

13.8.5 Examples of [XACML] Policies

The following examples illustrate how [XACML] Policies may be used to address come common use cases that have been identified.

13.8.5.1 Default Access Control Policy

The following Policy defines the default access control policy. This Policy must implicitly apply to any resource that does not have an explicit Acces Control Policy defined.
It consists of 3 rules, which in plain English are described as follows:
1. Any subject can perform read action on any resource
2. A subject may perform any action on a resource for which they have the role of ContentOwner.
3. A subject with role of RegistryAdministrator may perform any action on any resource.

The listing of the suggested default Access Control Policy follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:1.0:policy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
   C:\tmp\xacml\cs-xacml-schema-policy-01.xsd"
PolicySetId="urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:policy:default-access-
control-policy" PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-
combining-algorithm:permit-overrides">
   <Description>
```

```
    This PolicySet defines the default Access Control Policy for all registry
resources.
    </Description>
    <Target>
        <Subjects>
            <AnySubject/>
        </Subjects>
        <Resources>
            <AnyResource/>
        </Resources>
        <Actions>
            <AnyAction/>
        </Actions>
    </Target>
    <Policy PolicyId="urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:policy:policyid:permit-anyone-to-read"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides">
        <Target>
            <Subjects>
                <AnySubject/>
            </Subjects>
            <Resources>
                <AnyResource/>
            </Resources>
            <Actions>
                <AnyAction/>
            </Actions>
        </Target>
        <Rule RuleId="urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:rule:ruleid:permit-anyone-to-read" Effect="Permit">
            <Description>
            Any Subject can perform read action on any resource.
            </Description>
            <Target>
                <Subjects>
                    <AnySubject/>
                </Subjects>
                <Resources>
                    <AnyResource/>
                </Resources>
                <Actions>
                    <Action>
                        <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
```

```
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
                    <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </ActionMatch>
            </Action>
        </Actions>
      </Target>
    </Rule>
  </Policy>
  <Policy PolicyId="urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:policy:policyid:permit-owner-all"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
    <Rule RuleId="urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:rule:ruleid:permit-owner-all" Effect="Permit">
      <Description>
      A Subject with role of ContenOwner can perform any action on
resources owned by them.
      </Description>
      <Target>
        <Subjects>
          <AnySubject/>
        </Subjects>
        <Resources>
          <AnyResource/>
        </Resources>
        <Actions>
          <AnyAction/>
        </Actions>
      </Target>
      <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:anyURI-
equal">
```

```
                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:anyURI-
one-and-only">
                    <SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
                </Apply>
                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:anyURI-
one-and-only">
                    <ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:resource:owner"
DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
                </Apply>
            </Condition>
        </Rule>
    </Policy>
    <Policy PolicyId="urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:policy:policyid:permit-registryadministrator-all"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides">
        <Target>
            <Subjects>
                <AnySubject/>
            </Subjects>
            <Resources>
                <AnyResource/>
            </Resources>
            <Actions>
                <AnyAction/>
            </Actions>
        </Target>
        <Rule RuleId="urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:rule:ruleid:permit-registryadministrator-all" Effect="Permit">
            <Description>
            A Subject with role of RegistryAdministrator can perform any action on
any resource.
            </Description>
            <Target>
                <Subjects>
                    <Subject>
                        <SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">RegistryAdministrator<
/AttributeValue>
```

```
                                <SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                            </SubjectMatch>
                        </Subject>
                    </Subjects>
                    <Resources>
                        <AnyResource/>
                    </Resources>
                    <Actions>
                        <AnyAction/>
                    </Actions>
                </Target>
            </Rule>
        </Policy>
</PolicySet>
```

13.8.5.2 Custom Access Control Policy

The following Policy defines a custom access control policy to restrict *read access* to a resource to specified user or role. It also grants update access to specified role.
It consists of 3 rules, which in plain English are described as follows:

1. A subject may perform any action on a resource for which they have the role of ContentOwner. This reuses a Policy by reference from the default Access Control PolicySet.
2. A subject with role of RegistryAdministrator may perform any action on any resource. This reuses a Policy by reference from the default Access Control PolicySet.
3. A subject with specified id may perform read actions on the resource. This restricts read access to the specified subject.
4. A subject with role of Manager may perform update actions on the resource. This relaxes update access restrictions to the specified subject.

The listing of the custom Access Control Policy follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:1.0:policy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
    C:\tmp\xacml\cs-xacml-schema-policy-01.xsd"
PolicySetId="urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:policy:restricted-
access-control-policyset"
PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-
algorithm:permit-overrides">
```

```
    <Description>
    This PolicySet restricts the default Access Control Policy to limit read access
to specified subjects.
    </Description>
    <Target>
        <Subjects>
            <AnySubject/>
        </Subjects>
        <Resources>
            <AnyResource/>
        </Resources>
        <Actions>
            <AnyAction/>
        </Actions>
    </Target>
    <PolicyIdReference>urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:policy:policyid:permit-owner-all</PolicyIdReference>
    <PolicyIdReference>urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:policy:policyid:permit-registryadministrator-
all</PolicyIdReference>
    <Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
    C:\tmp\xacml\cs-xacml-schema-policy-01.xsd"
PolicyId="urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:policy:permit-delete-
access-control-policy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
combining-algorithm:permit-overrides">
        <Description>
        Allow Subject with specifed id to perform delete action on any resource.
        </Description>
        <Target>
            <Subjects>
                <AnySubject/>
            </Subjects>
            <Resources>
                <AnyResource/>
            </Resources>
            <Actions>
                <AnyAction/>
            </Actions>
        </Target>
        <Rule RuleId="urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:rule:ruleid:permit-delete-rule" Effect="Permit">
            <Description>
            Allow Subject with specifed id to perform delete action on any resource.
        </Description>
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#anyURI">urn:uuid:977d9380-
00e2-4ce8-9cdc-d8bf6a4157be</AttributeValue>
                        <SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
                    </SubjectMatch>
                </Subject>
            </Subjects>
            <Resources>
                <AnyResource/>
            </Resources>
            <Actions>
                <Action>
                    <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">delete</AttributeValue
>
                        <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
                    </ActionMatch>
                </Action>
            </Actions>
        </Target>
    </Rule>
</Policy>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
    C:\tmp\xacml\cs-xacml-schema-policy-01.xsd"
PolicyId="urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:policy:permit-update-
access-control-policy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
combining-algorithm:permit-overrides">
    <Description>
    Allow Subjects with Manager role to perform update action on any
resource.
    </Description>
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
    <Rule RuleId="urn:oasis:names:tc:ebxml-
regrep:2.5:rim:acp:rule:ruleid:permit-update-rule" Effect="Permit">
      <Description>
      Allow Subjects with Manager role to perform read action on any
resource.
      </Description>
      <Target>
        <Subjects>
          <Subject>
            <SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Manager</AttributeVal
ue>
              <SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:ebxml-regrep:2.5:rim:acp:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </SubjectMatch>
          </Subject>
        </Subjects>
        <Resources>
          <AnyResource/>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">update</AttributeValue
>
              <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </ActionMatch>
```

```
                    </Action>
                </Actions>
            </Target>
        </Rule>
    </Policy>
</PolicySet>
```

13.8.6 Resolving PolicyReferences

An XACML PolicySet may reference XACML Policy objects defined outside the repository item containing the XACML PolicySet. A registry implementation must be able to resolve such references. To resolve such references efficiently a registry must be able to find the repository item containing the referenced Policy without having to load and search all Access Control Policies in the repository. This section describes the normative behaviour that enables a registry to resolve policy references efficiently.

A registry must define a Content Cataloging Service for the canonical XACML PolicySet objectType. The PolicySet cataloging service must automatically catalog every PolicySet upon submision to contain a special Slot with name “ComposedPolicies”. The value of this Slot must be a Collection where each element in the Collection is the id for a Policy object that is composed within the PolicySet.

Thus a registry is able to use an ad hoc query to find the repositoryItem representing an XACML PolicySet that contains the Policy that is being referenced by another PolicySet.

13.8.7 ebXML Registry as a [XACML] Policy Store

So far we have defined how ebXML registries may use [XACML] to define Access Control Policies to control access to RegiostryObejct and RepositoryItem resources.

An important side effect of the normative binding of the Access Control Model to [XACML] is that enterprises may also use ebXML Registry as a [XACML] Policy store to manage Policies for protecting resources outside the registry.

In this use case, enterprises may submit [XACML] Policies and PolicySets as ExtrinsicObject-RepositoryItem pairs. These Policies may be accessed or referenced by their URL as defined by the HTTP binding of the ebXML Registry Services interface in [ebRS].

13.9 Access Control Model: Custom Binding

A registry may support bindings to policies describes in formats other than [XACML]. The use of such policies sacrifices interoperability and is therefore discouraged. In such cases the RepositoryItem for the policy information may be in any format supported by the registry in an implementation specific manner.

Appendix A Canonical Classification Schemes

This section lists the canonical ClassificationSchemes that are required to be present in all ebXML Registries. These Canonical ClassificationSchemes may be extended by adding additional ClassificationNodes. However, no ClassificationNode defined normatively in the links below may be modified within a registry. In particular they must preserve their canonical id attributes in all registries.

A.1 ObjectType ClassificationScheme

http://www.oasis-open.org/committees/regrep/documents/2.5/canonical/SubmitObjectsRequest_ObjectTypeScheme.xml

A.2 AssociationType ClassificationScheme

http://www.oasis-open.org/committees/regrep/documents/2.5/canonical/SubmitObjectsRequest_AssociationTypeScheme.xml

A.3 PhoneType ClassificationScheme

http://www.oasis-open.org/committees/regrep/documents/2.5/canonical/SubmitObjectsRequest_PhoneTypeScheme.xml

A.4 EmailType ClassificationScheme

http://www.oasis-open.org/committees/regrep/documents/2.5/canonical/SubmitObjectsRequest_EmailTypeScheme.xml

A.5 ContentManagementService ClassificationScheme

http://www.oasis-open.org/committees/regrep/documents/2.5/canonical/SubmitObjectsRequest_CMSScheme.xml

A.6 ErrorHandlingModel ClassificationScheme

http://www.oasis-open.org/committees/regrep/documents/2.5/canonical/SubmitObjectsRequest_ErrorHandlingModelScheme.xml

A.7 InvocationModel ClassificationScheme

http://www.oasis-open.org/committees/regrep/documents/2.5/canonical/SubmitObjectsRequest_InvocationModelScheme.xml

A.8 SubjectRole ClassificationScheme

http://www.oasis-open.org/committees/regrep/documents/2.5/canonical/SubmitObjectsRequest_SubjectRoleScheme.xml

A.9 SubjectGroup ClassificationScheme

http://www.oasis-open.org/committees/regrep/documents/2.5/canonical/SubmitObjectsRequest_SubjectGroupScheme.xml

14 References

[ebGLOSS] ebXML Glossary,
http://www.ebxml.org/documents/199909/terms_of_reference.htm

[OAS] OASIS Information Model
http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf

[ISO] ISO 11179 Information Model
http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument

[BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use in RFCs to Indicate Requirement Levels
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html

[ebRS] ebXML Registry Services Specification
http://www.oasisopen.org/committees/regrep/documents/2.5/specs/ebRS.pdf

[ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
http://www.ebxml.org/specfrafts/

[UUID] DCE 128 bit Universal Unique Identifier
http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
http://www.opengroup.org/publications/catalog/c706.htmttp://www.w3.org/TR/REC-xml

[XPATH] XML Path Language (XPath) Version 1.0
http://www.w3.org/TR/xpath

[XACML] OASIS eXtensible Access Control Markup Language (XACML) Version 1.0
http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-01.pdf

[NCName] Namespaces in XML 19990114
http://www.w3.org/TR/REC-xml-names/#NT-NCName.

15 Disclaimer

The views and specification expressed in this document are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this design.

16 Contact Information

Team Leader

Name:	Kathryn R. Breininger
Company:	The Boeing Company
Street:	P.O. Box 3707 MC 62-LC
City, State, Postal Code:	Seattle, WA 98124-2207
Country:	USA
Phone:	425-965-0182
Email:	kathryn.r.breininger@boeing.com

Editor

Name:	Sally Fuger
Company:	Automotive Industry Action Group
Street:	26200 Lahser Road, Suite 200
City, State, Postal Code:	Southfield, MI 48034
Country:	USA
Phone:	(248) 358-9744
Email:	sfuger@aiag.org

Technical Editor

Name:	Farrukh S. Najmi
Company:	Sun Microsystems
Street:	1 Network Dr., MS BUR02-302
City, State, Postal Code:	Burlington, MA, 01803-0902
Country:	USA
Phone:	(781) 442-9017
Email:	farrukh.najmi@sun.com

Technical Editor

Name:	Nikola Stojanovic
Company:	Metaspaces Consulting Street:
101 Pineview Terrace	
City, State, Postal Code:	Ithaca, NY, 14850
Country:	USA
Phone:	(607) 273-2224
Email:	nikola.stojanovic@acm.org

17 Copyright Statement