OASIS ebXML Registry

Proposal: Support for Extramural Associations

Category: Improvements to existing specifications

Date: November 1, 2001

Author: Farrukh Najmi

Status of this Document

This document is a draft proposal whose purpose is to solicit additional input.

1 Abstract

The RIM 1.1 specification defines an Association class to represent an association between two RegistryObjects. Current definition of association has the following limitations:

1. An Association is restricted to be owned by the owner of the sourceObject in the Association.
2. The rules are not clearly defined when the sourceObject and targetObject of the Association are owned by different submitting organizations (SO).
3. There is no support for bilaterally agreed upon associations between two objects that are owned by different submitting organizations.

This document proposes to provide focused backward compatible enhancements to RIM1.1 and RS 1.0 that provide solutions to the above problems.

2 Motivation

The following motivations drive this proposal:

1. Clarify current specifications for the case of extramural Associations
2. Relax restrictions on Associations

2.1 Assumptions

The following assumptions are made in this proposal:

1. Issues dealing with multiple co-operating registries are not considered. These issues are deferred to the Inter Registry Cooperation (IRC) team.

3 Changes to RIM 1.1

Replace chapter 9 with following chapter. A replacement of the chapter is needed because we need to incrementally expose complexity to the reader and properly position intramural Vs. extramural associations.

4 Association of Registry Objects

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.

4.1 Example of an Association

One example of such an association is between two ClassificationScheme instances, where one ClassificationScheme supercedes the other ClassificationScheme as shown in Figure 1. This may be the case when a new version of a ClassificationScheme is submitted.

In Figure 1, we see how an Association is defined between a new version of the NAICS ClassificationScheme and an older version of the NAICS ClassificationScheme.

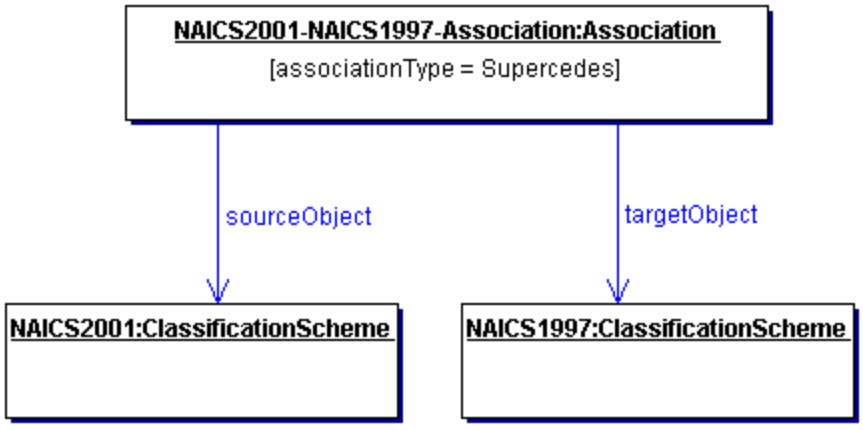

Figure 1: Example of RegistryObject Association

4.2 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 1, 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 supercedes the targetObject (and not the other way around).

4.3 Association Types

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

4.4 Intramural Associations

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 same User that is defining the Association. Such associations are referred to as *intramural associations*.

Figure 2 below, extends the previous example in Figure 1 for the intramural association case.

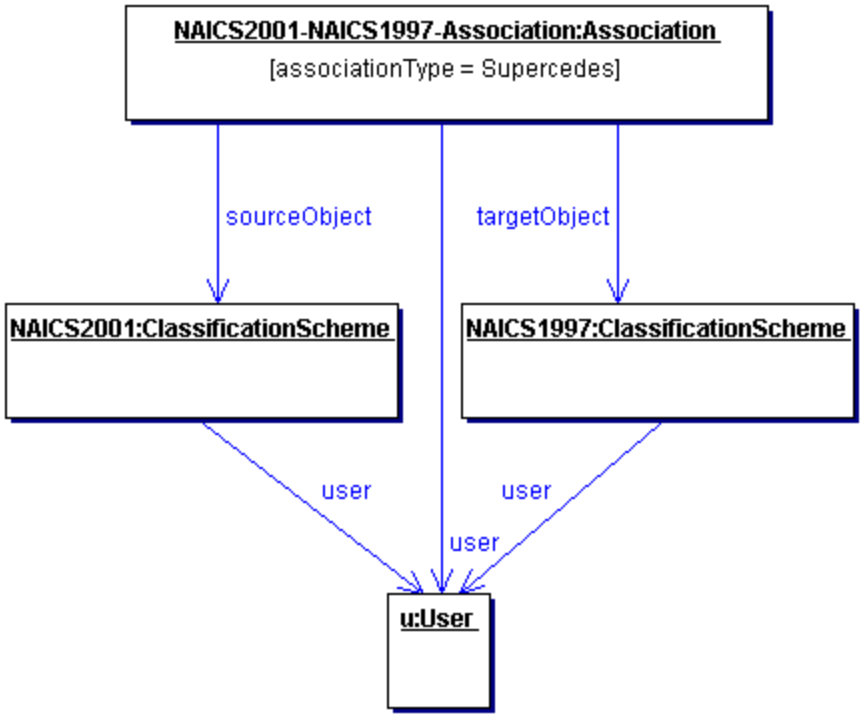

Figure 2: Example of Intramural Association

4.5 Extramural Association

The information model also allows a more sophisticated use case where 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 the Association is being 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 3 below, extends the previous example in Figure 1 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 3. In such case, one of the two users owns two of the three objects involved (Association, sourceObject and targetObject).

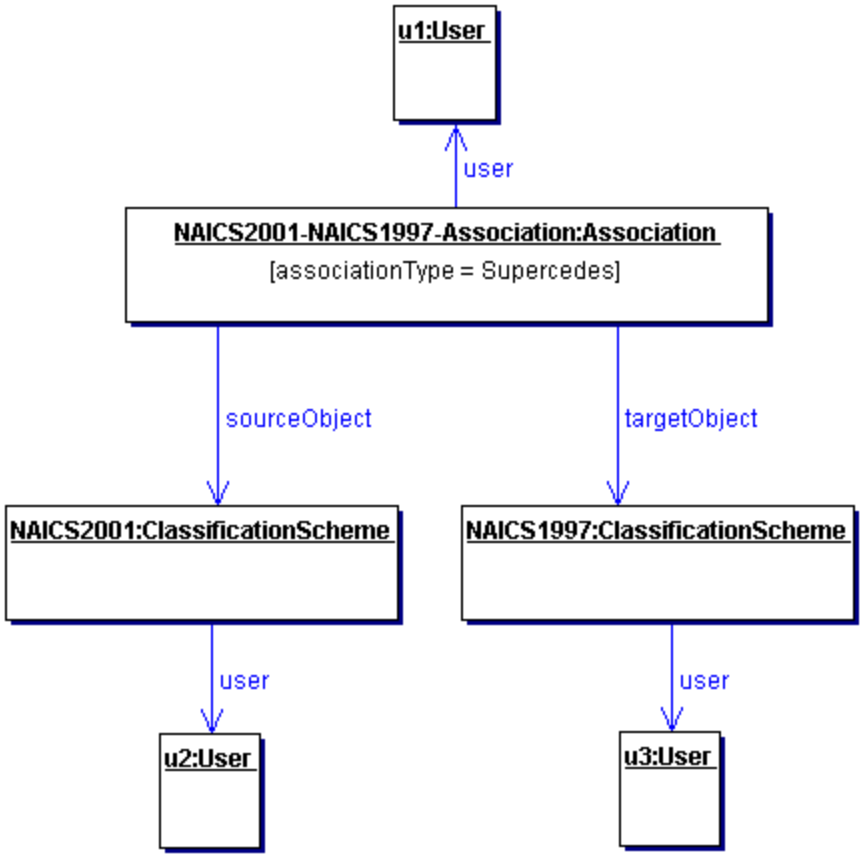

Figure 3: Example of Extramural Association

4.6 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.

4.6.1 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.

4.6.2 Confirmation of Extramural Associations

Extramural associations may be viewed as a unilateral assertion that may not be viewed as truth until it has been confirmed by the other (extramural) parties (Users “u2” and “u3” in example in section 4.5).

To confirm an extramural Association, each extramural parties (parties that own source or target object that 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 have the same id as the original association.

4.7 Visibility of Unconfirmed Associations

Extramural associations require each extramural party to confirm the assertion being made by the extramural Association before the Association is visible to 3rd parties that are not involved in the Association. This ensures that unconfirmed Associations are not visible to 3rd party registry clients.

4.8 Possible Confirmation States

Assume the most general case where there are three distinct User instances as shown in Figure 3 for an extramural Association. The extramural Association needs to be confirmed by both the other (extramural) parties (Users “u2” and “u3” in example) in order to be fully confirmed. The methods `isConfirmedBySourceOwner` and `isConfiremedByTargetOwner` in the Association class provide access to confirmation state for both the sourceObject and targetObject. A third 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 confirmation state of an extramural Association:

- The Association is confirmed neither by the owner of the sourceObject nor is it confirmed by owner of targetObject.
- The Association is confirmed by the owner of the sourceObject but it is not confirmed by owner of targetObject.
- The Association is not confirmed by the owner of the sourceObject but it is confirmed by owner of targetObject.
- The Association is confirmed by the owner of the sourceObject and it is confirmed by owner of targetObject. This is the only state where the Association is fully confirmed.

4.9 Class *Association*

Super Classes:

RegistryObject

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

An *Instance* of the Association *Class* represents an association between two RegistryObjects.

4.9.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
associationType	LongName	Yes		Client	No
sourceObject	UUID	Yes		Client	No
targetObject	UUID	Yes		Client	No

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

4.9.2 Attribute associationType

Each Association must have an associationType attribute that identifies the type of that association. This MUST be the name attribute of an association type as defined by **Error! Reference source not found.**.

4.9.2.1 Pre-defined Association Types

The following table lists pre-defined association types. These pre-defined association types are defined as a *Classification* scheme. While the scheme may easily be extended a *Registry* MUST support the association types listed below.

name	description

RelatedTo	Defines that source RegistryObject is related to target RegistryObject.
HasMember	Defines that the source Package object has the target RegistryObject object as a member. Reserved for use in Packaging of RegistryEntries.
ExternallyLinks	Defines that the source ExternalLink object externally links the target RegistryObject object. Reserved for use in associating ExternalLinks with RegistryEntries.
Contains	Defines that source RegistryObject contains the target RegistryObject.
EquivalentTo	Defines that source RegistryObject is equivalent to the target RegistryObject.
Extends	Defines that source RegistryObject inherits from or specializes the target RegistryObject.
Implements	Defines that source RegistryObject implements the functionality defined by the target RegistryObject.
InstanceOf	Defines that source RegistryObject is an *Instance* of target RegistryObject.
Supersedes	Defines that the source RegistryObject supersedes the target RegistryObject.
Uses	Defines that the source RegistryObject uses the target RegistryObject in some manner.
Replaces	Defines that the source RegistryObject replaces the target RegistryObject in some manner.

158

159 4.9.3 Attribute sourceObject

160 Each Association must have a sourceObject attribute that references the
161 RegistryObject instance that is the source or owner of that association.

162 4.9.4 Attribute targetObject

163 Each Association must have an targetObject attribute that references the
164 RegistryObject instance that is the target of that association.

4.9.5 Inherited Attribute id

The id attribute for an Association is an attribute based id composed of the value of the sourceObject, targetObject and associationType attributes in that order, where each attribute value is separated by a ':'.

The pattern is as follows:

urn:uuid:< sourceObject id>:< targetObject id>:<associationType>

An example is as follows:

urn:uuid:a2345678-1234-1234-123456789012: a2345678-1234-1234-123456789013:Implements

Method Summary of Association	
boolean	**isConfirmed**() Returns true if isConfirmedBySourceOwner and isConfirmedByTargetOwner both return true. For intramural Associations always return true. An association should only be visible to third parties (not involved with the Association) if isConfirmed returns true.
boolean	**isConfirmedBySourceOwner**() Returns true if the association has been confirmed by the owner of the sourceObject. For intramural Associations always return true.
boolean	**isConfirmedByTargetOwner**() Returns true if the association has been confirmed by the owner of the targetObject. For intramural Associations always return 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.