OASIS ebXML Registry Technical Committee

Subject: AuditableEvent: Issues re: ServiceRequests and Mappings
Version: 1
Author: Len Gallagher
Date: 24 October, 2001
Type: ebRIM Issue Paper with Proposals

Introduction

This is an issues paper that focuses on Auditable Event, an important class in the model. It is my opinion that AuditableEvent is under-specified in ebRIM and that it may be necessary to create a new class, e.g. ServiceRequest, in order to satisfy known Registry requirements.

I see audit Registry requirements including at least the following:

1) A Registry needs to keep an audit trail of every service request that has any effect on persistent values stored in the Registry. Thus we need an audit log of every SubmitObjectsRequest, ApproveObjectsRequest, DeprecateObjectsRequest, or RemoveObjectsRequest that is received and acted upon by Registry Services.

2) A Registry needs to be able to determine which ServiceRequests have an impact on which RegistryObjects. In particular, a client needs to be able to ask for an Audit Trail of ServiceRequests that had an impact on a given RegistryEntry.

I think our current specification of AuditableEvent fails these requirements in several ways, including the following observations from Figure 1 in ebRIM v1.1:

1) We don't see any class to identify ServiceRequests as independent instances. Thus it seems impossible to keep a record of ServiceRequests that delete objects from the Registry. For example, if I submit a RemoveObjectsRequest to delete an Association instance, I don't see any way to record that information because the Association instance will no longer be in the Registry. There are no rules for how I might record the deletion as an impact on the RegistryEntry that is the sourceObject or targetObject of the Association instance.

2) The relationship from AuditableEvent to RegistryObject labeled as "registryObject" implies that there is at most one RegistryObject instance that is impacted by that AuditableEvent instance. But a SubmitsObjectsRequest may submit thousands of objects at one time, and each individual submission, i.e. a new Association instance, may impact both the source and target objects of the association. Similarly, if a new object is submitted to supersede an existing object, the RegistryEntry of both the old object and the new object will be affected.

I think we can address the above limitations of our existing specification in the following way:

1) Create a new class, call it ServiceRequest, to keep a record of every service request received and acted upon by Registry Services. Require that each ServiceRequest instance be timestamped and linked to the User instance who submitted it.

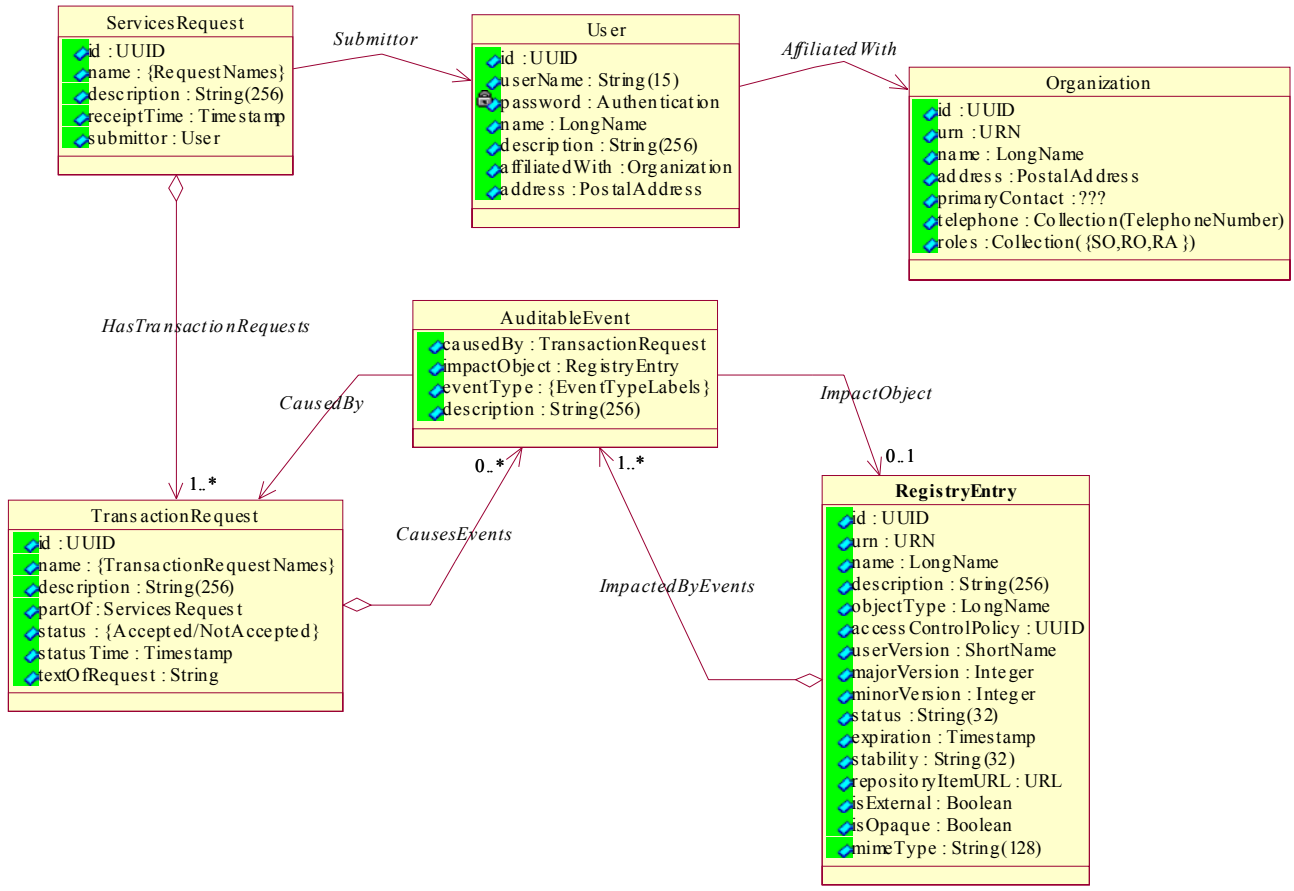
2) Since some service requests may consist of many smaller requests, e.g. a SubmitObjectsRequest many actually submit thousands of new objects, consider keeping track of requests at a lower level of granularity. For example, consider identifying each new object submitted as part of a SubmitObjectsRequest to be an independent request on Registry Services that can be accepted or rejected. If accepted, we need to record its impact via an AuditableEvent on other objects in the Registry, or if rejected, we need to record that the request was received but rejected for some reason. Call this new class a TransactionRequest because at this level of granularity each request will satisfy the ACID properties of a transaction; in particular it will be totally successful or totally rejected. Require that every TransactionRequest be linked to its parent ServiceRequest so that a User and timestamp can be inferred, and provide an attribute to record whether the request was accepted or rejected. Possibly, even provide an attribute to retain (for a limited audit period)

the text of the entire request so that potential disputes between Client and Server can be addressed in detail. Afetr the expiration of the agreed audit period the actual text of the request could be deleted, but NOT the existence of the request itself.

3) With the above new classes, let AuditableEvent represent a many-to-many mapping between TransactionRequest and RegistryEntry. Note that I choose RegistryEntry here instead of RegistryObject. This is because I feel it is impractical to retain an audit trail on every action on every object in the Registry, especially when deletions need to be recorded. Instead, we only need to track actions that have an impact on a "repository item". And impacts on "repository items" can be represented by impacts on the RegistryEntry that describes that "repository item".

Please consider the following diagram as a possible submodel of ebRIM for describing Audit Requirements.

Audit Trail Submodel



{RequestNames}={SubmitObjectsRequest, ApproveObjectsRequest, DeprecateObjectsRequest, RemoveObjectsRequest, etc.}

{TransactionRequestNames}={SubmitObject, ApproveObject, DeprecateObject, RemoveObject, etc.}

{EventTypeLabels}={AddAssoc, DropAssoc, RefByAssoc, AddClassif, DropClassif, AddExtId, DropExtId, AddExtLink, DropExtLink, ChgStatus, etc.}

Outline Of a Proposal

Make the following modifications to Section 7, "Registry Audit Trail", of ebRIM v1.1

1) Add ServicesRequest as a new RIM class, with the attributes defined in the UML diagram, and with methods that correspond to the pictured UML relationships.

2) Add TransactionRequest as a new RIM class, with the attributes defined in the UML diagram, and with methods that correspond to the pictured UML relationships.

3) In Section 7.1, "Class AuditableEvent", remove the sentence that says AuditableEvent is a subtype of RegistryObject. Instead, it becomes a stand-alone class representing labeled many-to-many associations between TransactionRequest and RegistryEntry.

4) In Sections 7.1.x, Attributes of AuditableEvent, modify the attribute summary and the attribute definitions to match the attributes from the above UML diagram.

5) Add a new Section 7.1.x, "Method Summary of AuditableEvent", with two methods as follows:
   getTransactionRequest():TransactionRequest (returns the whole instance, not just a UUID)
   getRegistryEntry():RegistryEntry (returns the whole instance, not just a UUID)

6) Add the above UML diagram to the introduction of Section 7, and call it a detailed Audit Trail sub-model of Figure 1 (ebRIM page 11).

7) Modify Figure 1 (ebRIM page 11) to make it consistent with the above UML diagram. Alternatively, delete AuditableEvent and User from Figure 1 so that their specification can come completely in Section 7.

8) Consider adding ServicesRequest and TransactionRequest to Figure 2 (ebRIM page 14) as sub-classes of RegistryObject.

9) Remove AuditableEvent from Figure 2 (ebRIM page 11) because it no longer has "id" or "name" attributes.

10) Remove getAuditableEvents() as a method on RegistryObject (ebRIM Section 6.3.7). Instead, define new methods on RegistryEntry (ebRIM Section 6.4.x) as follows:
    getAuditableEvents(String:eventType):Collection(AuditableEvent) (returns set of whole instances)
    getEventTransactions(String:eventType):Collection(TransactionRequest) (returns UUID's is OK).