A JPA Framework for Integrated Collaboration Environments

Part 1

Eric S. Chan
Oracle Corporation
March 9, 2011

Abstract

This article is the first part of the three parts series to present a new project in java.net (http://java.net/projects/open-icom) to incubate a JPA framework for developing integrated collaboration environments. The first part explains the advantages of the JPA programming model, which embodies the design patterns that are well-suited for managing integrated collaboration object model (ICOM). ICOM is a framework defined by the OASIS ICOM Technical Committee to integrate a broad range of domain models for a collaboration environment. ICOM JPA framework will lower the barrier for application developers to develop collaboration tools to support seamless transitions across collaboration activities with minimal context switching. It will encourage independent software vendors and open source communities to create common collaboration clients that interoperate with integrated collaboration platforms and standalone collaboration services across enterprise boundaries. The article provides an overview of ICOM with programming examples for the ICOM JPA framework. It covers the high-level concepts, directory, space, access control, metadata, content management, and unified message model. The second part of the series will present additional extension modules defined in the ICOM specification. The design of the ICOM JPA framework will be described in the third part of the series.

Introduction

An increasing number of application APIs are standardized on Java Persistence API[1][2], which is a JSR specification that incorporates the proven solutions from leading open source and commercial object-relational mapping (ORM) frameworks, including Hibernate and TopLink. JPA is an interoperability standard that affords developers greater choices of technologies, including the open ORM frameworks as well as proprietary implementations, without subjecting the developers to a lowest common denominator.

Although JPA is an API for management of persistence and object-relational mapping, its embodiment of the general information management design patterns, such as portable POJO domain model, delineation of managed and detached objects, in-memory transactions, eager and lazy loading of states, attribute level change tracking, cascade persist (a relatively newer notion than the cascade delete notion), persistence context and second-level caches, and object query language, makes it attractive for

---

1 ICOM specification is currently a working draft in OASIS ICOM TC workspace http://www.oasis-open.org/apps/org/workgroup/icom.
a broader array of information management domains that may not be implemented entirely by ORM. An infrastructure to support these information management capabilities will relieve the application developers from encumbrances of writing the plumbing codes so that they can instead focus on their business domains.

One such information management domain that can benefit from a common infrastructure based on JPA programming model is the integrated collaboration environment. The OASIS Integrated Collaboration Object Model for Interoperable Collaboration Services Technical Committee (ICOM TC)[3] is defining the normative standards for collaboration objects, along with the classes, attributes, relationships, constraints, and behavior, for an integrated and interoperable collaboration environment. The specification is intended for integrating a broad range of collaboration objects to enable seamless transitions across collaboration activities. This enables applications to provide continuity of conversations across multiple collaboration channels. For example, applications can aggregate conversation threads in email with other conversations on the same topic in instant message, over the phone or via real-time conferencing, by discussion threads in community forum, wiki, weblog or micro blog, and activity stream of participants from all channels.

The JPA infrastructure for ICOM provides a unified programming model as a frontend for a set of existing protocols/services that involve several domain models from disparate technologies such as LDAP, JCR, IMAP, SMTP, XMPP, iCalendar, CalDAV, WebDAV, vCard, FOAF, SIOC, Facebook Open Graph, OpenSocial, BPEL, BPEL4People, etc. In addition to unifying the domain model, JPA offers the infrastructure for lazy loading, change tracking, attach/detach/merge, cascade persist, L1 and L2 cache, etc, capabilities that are disjointed or non-existent in the existing protocols/services. To accommodate third-party integrated collaboration platforms as well as discrete protocols/services, the ICOM JPA framework supports pluggable data access connectors. A data access connector can be implemented using the proprietary API for a vendor’s integrated collaboration platform or composed from the DAO components for discrete collaboration and content management services. Figure 1 shows a few example DAO components comprising a data access connector.
The JPA infrastructure for collaboration technologies will be an important addition to the Java technology. We call on the java.net community to contribute data access connectors or DAO components for third-party collaboration and content management services. The framework includes a prototype data access connector for Oracle Beehive Collaboration Platform using a proprietary collaboration service interface. An alternative data access connector can be implemented using the REST/SOAP interfaces[4]. To support a JPA level-2 shared cache for a multi-user environment, the ICOM JPA framework will need to be extended to support access control enforcement. ICOM standardization of access control model makes it feasible to implement such a second-level cache extension in server environments.

A JPA framework that uses a federation of data access connectors, which themselves can be composed of discrete data access objects through separate service protocols, with various levels of transaction support by the services, inevitably compromises the ACID transaction properties. The reader is referred to the experience and usability studies[5] of the weaker consistency properties, ranging from Eventual Consistency to single-entity ACID, of Not Only SQL (NoSQL) cloud storage systems. The NoSQL cloud services such as Amazon S3 and SimpleDB, Google Datastore, Microsoft Azure Storage, and Cassandra relax the data consistency requirements to achieve throughput, availability, and elasticity requirements.
**Breaking down the barriers**

Enterprises have been breaking down the walls between the internal organizations to get people to collaborate across stovepipes\[6\] sometimes by flattening the organizational structures and other times by deploying technologies such as team workspaces, forums, wikis, etc., that promote cross-department collaboration. They have also been opening up their enterprise information systems for coordination of product specifications, engineering drawings, computer-assisted design tools, order fulfillment, procurement, billing, inventory and delivery tracking, logistics, etc., with business partners to streamline the supply-chains. An empirical study\[7\] showed that these e-collaboration tools improved process innovation and performance of the supply chains and offered greater process flexibility for upstream partners in the supply chains. However, they still fall short of empowering the employees to collaborate across enterprise boundaries with agility to react to exceptions in the business processes, a capability which is required to extend benefits for downstream and upstream partners in the supply chains. Nowadays, enterprises are increasingly opening up and collaborating with external organizations on larger and specialized projects for the mutual benefit of all partners, sometimes by forming dynamic virtual teams. Boeing 787 project was an example where an old supply-chain gave way to a new value-network of partners who share information through inter-enterprise collaborative workspaces in a seamless community\[13\].

Organizations have incrementally deployed a mix of disjoint collaboration tools. The increasingly fragmented tools only erode the productivity. The fragmented collaboration tools are usually technology driven tools that require constant context switching for the users to perform a single task. The fragmentation leads to incomplete threads of conversations when users communicate through multiple tools. The silos of tool repositories prevent the users from relating, aggregating, and reasoning about diverse types of collaboration artifacts by project, task, metadata, or any relevant context. The data silos also prevent uniform relevance rankings of search results from the isolated repositories. The proliferation of web 2.0 content silos also weakens corporate governance. On the other hand the enterprises need to integrate the artifacts that the employees generate in the unstructured collaboration activities with the enterprise business objects of the structured business processes. They need a standardized API and model for developing composite applications for contextual collaboration and semi-structured, project-centric collaboration processes where unstructured collaboration activities intersect with structured business processes.

Many organizations are faced with the technical obstacles and high costs in their quests to integrate the disjoint tools and the silos of data each tool produces. Projects to integrate the silos of repositories encountered the soaring costs or technical barriers. To solve the fragmentation problem, various collaboration vendors have attempted to unify their platforms in order to build a single collaboration environment which provides the full range of collaboration activities. However, these vendor specific platforms still lack a standard model, interface, and protocol to support contextual collaboration within business processes. Without a standard collaboration model that can provide a complete range of collaboration activities, customers, independent software vendors, and system integrators face a difficult challenge to build contextual collaboration environments using service components from
multiple vendors. OASIS ICOM TC was chartered two years ago to define a standard collaboration model to address the integration challenges. ICOM covers a broad, extensible range of collaboration activities encompassing, and in some cases improving on, a range of models in existing standards and technologies that were developed independently and had created the impedances between the components across standard and technology boundaries.

The walls between internal organizations and across enterprises have been coming down with the deployment of cross-boundary collaboration technologies. One of the technologies is the social networking service which let a user create a public or semi-public profile in an articulated network of profiles to share experience through activity streams, blogs in friends’ dashboards, ratings and recommendations, and other shared media. Gartner report[10] projected that social networking tools will replace email as primary interpersonal communication tool for 20 percent of business users by 2014. In the consumer world, social networking activities have already surpassed email as the most popular online activities. Some external facing employees in marketing, sales, and customer care, have to use popular social networking sites, such as Facebook, Twitter, YouTube, etc., to interact with consumers. Integrated collaboration environments should provide an integration point to the consumer social networking sites to offer more open experience. The JPA infrastructure for ICOM can support data access objects for Open Graph and OpenSocial APIs[11][12] to integrate with consumer social networking sites.

ICOM can further accelerate the removal of the walls between the collaboration tools and also expose the data from behind the wall of applications. Exposing the data in machine readable form is what Semantic Web is about. OASIS ICOM TC includes representations from Digital Enterprise Research Institute (DERI) and Ontolog Forum, whose focus areas are in semantics technologies. ICOM ontology is defined from the outset for concomitant representation in UML and RDF. ICOM TC wiki page[3] discusses the mappings between UML and RDF representations. ICOM can bridge the object-oriented software engineering world with the semantic web world by providing bi-directional transformations between UML and RDF. Linked Data Community[8] is advancing the use of web, URI, and RDF to connect distributed data. There is a popular saying “a little semantic goes a long way” about enriching the data with inference capability. ICOM data with a seamless programming model like JPA and a concomitant RDF representation will lower the barrier for applying inference engines such as JESS, OWL, and SPARQL. Figuratively speaking, a rich vocabulary of “nouns” in ICOM makes up for the strong “verbs” in service interfaces. A well-defined set of classes of ICOM makes the API amenable for rule-based applications and declarative inference. ICOM containers are active or reactive entities, for example conference and chat rooms are highly active while outbox, calendar, and task list are reactive. Their behavior can be augmented by applications.

**Overview of ICOM**

ICOM specification defines a class called Entity which is the super class of any class that supports a persistent identifier, a change token for optimistic locking, and an access control list. The object identifier and change token are annotated, respectively, by “javax.persistence.Id” and
“javax.persistence.Version,” matching the ICOM concept of Entity with the JPA concept of Entity. ICOM Entity has another fundamental dimension for access control list, which together with JPA Id and Version, defines a unit of persistent information for concurrency and access control. The generation of object identifiers is implementation dependent; however, ICOM recommends that the object identifiers should be globally unique to support permanent references to the entities that may migrate amongst interoperable ICOM repositories. An object identifier is read only (immutable) once it is assigned and should never be duplicated or re-used for more than one object. The UML diagram in Figure 2 depicts the Entity class, properties, and cardinality of the properties. Entity’s properties include name, created by, creation date, last modified by, last modification date, owner, parent, attached markers, category applications, tag applications, and access control list.

![UML Class Diagram for Entity](image)

Figure 2 UML Class Diagram for Entity.

Here we explain the usage of each of the properties of Entity shown in Figure 2. ICOM relies on object identifier for persistent identification of an entity and allows the name string of the entity as an optional property. The actor who creates an entity is represented by the created by attribute. The created by and creation date attributes are set by the system and cannot be changed once they are set, hence created by and creation date are read-only attributes for applications. The last modified by and last modification date attributes can be updated by applications although ordinarily these attributes should be set by the system. The owner of an entity can be either a single actor or a group of actors. The parent of an entity can be a container, an extent, or a parental entity. The attached markers attribute of an entity represents the category and tag metadata entities associated with the entity. The category applications or tag applications attributes of an entity represent the instances of associations between categories or tags with the entity. These two attributes can also represent the multiple instances of associations between a category or tag with an entity, for example when different users apply the same tag on an
entity, each instance is represented by a different tag application. Access control list of entity provides fine grained access control policy at individual entity level.

An access control list (ACL) is an object attached to an entity to specify a list of permissions to access the entity. See Figure 3. The subject of an access control entry is an Accessor, which includes Group and Actor.

![UML Class Diagram of Access Control List](image)

**Figure 3 UML Class Diagram of Access Control List.**

The UML class diagram in Figure 4 depicts the top level classes of ICOM. The five subclasses of Entity, namely Relationship, EntityDefinition, Scope, Subject, and Artifact, inherit the properties from Entity. Of these five subclasses, Relationship and EntityDefinition are metadata concepts. Scope, Subject, and Artifact represent the forking of three major branches of ICOM.
Parental, Extent, and Container are mixin classes depicted as Java interfaces in Figure 4. The parental concept avoids overloading two aspects of parent-child relationship on Extent or Container. There are a few classes in ICOM that are Parental but not Extent or Container. A unified message can be a parent of an attachment, such as a forwarded message, replied to message, or document. Similarly, a document can be a parent of parts of the document. A parent-child relationship between a unified message and its attachments is different from a parent-child relation between an extent or container and its elements: the elements of an extent or container can have their own ACLs to override the ACL of the extent or container; on the other hand, attachments in the unified message always share the same ACL with the parent message, such that if a user can read the message, he or she can also read the attachments in the message. UnifiedMessage and Document are defined as Parental instead of Extent or Container. Actor, which can be a parent of user credential’s principals, pseudonyms, avatars, etc., is also defined as Parental. Community is defined as Extent while Space and Folder are defined as Container. See Figure 5.
The three major branches of ICOM Entity are described below.

**Scope Branch**
Scope includes Community and Space as shown in Figure 6.

Scope: A scope is an extent of a role assignment such that the privileges of the role are applicable only for operations on the entities in the scope. A scope contains role definitions, roles, and groups. A scope is relationship bondable so has a property for a set of relationships of a scope. See Figure 7.
Community: A community is a scope that contains sub-communities, spaces, actors, groups, roles, and role definitions. See Figure 8. It is implementation-dependent whether or not a space in a community can include participating actors who are not members of a parent community or ancestor communities.

Space: A space is a scope that defines a durable context and place for actors to work or collaborate. A space contains space items, which include sub-containers such as inbox, calendar, task list, conference, chat room, library, to name a few. See Figure 9.
Subject Branch

Subject branch includes Role, Group, and Actor as shown in Figure 10.

Subject: A subject is an entity who can have rights to perform actions on entities. Subject is relationship bondable so has a property for a set of relationships of a subject. Subject can have an extensible list of properties. See Figure 11.

Role: A role assigns a named set of privileges to a set of accessors for operations within an assigned scope. A role definition associated with a role represents a named set of privileges. See Figure 12. The role definitions can be carefully designed with minimal sets of privileges needed to perform specific
tasks. These role definitions can be created in the higher-level scopes to be used in assigning roles at the lower-level scopes. This practice promotes the principle of least privileges and uniformity of roles across spaces of a community. The role definitions can be part of the templates for creating new project spaces. For example, an implementation can supply a role definition for minimal privileges needed to coordinate the memberships of a project space. This role definition can be used to assign space coordinator role to any new project space. The privileges assigned by the role are applicable only for operations on entities in an assigned scope. A role is listed as a member of one or more communities or spaces.

![Figure 12 UML Class Diagram for Role.](image)

Group: A group is a named set of actors that can be assigned to a role, assigned as a subject in an access control list, and used as a buddy list, email distribution list, or participant list in collaboration activities. A group’s properties include member actors, member groups, assigned groups, assigned roles, assigned scopes, addresses, and primary address. A group is listed as a member of one or more roles, communities, or spaces. It is addressable and can be an owner of one or more entities. See Figure 13.

Actor: An actor can act on entities according to the access rights that are derived from assigned roles and access control lists where it appears as a subject. An actor’s properties include assigned groups, assigned roles, assigned communities, addresses, and primary address. An actor is listed as a member of one or more groups, roles, or communities. It is addressable and can be an owner of one or more entities. See Figure 13.
The classes in Scope and Subject branches, including Community, Space, Role, Group, and Actor, are components for representing the Directory and Membership constructs as described below.

Directory: A directory is hierarchical classified listings of Role, Group, and Actor for administration, search and indexing, and uniform reference. ICOM model for hierarchy of communities represents the directory features similar to LDAP schema. A community’s properties for roles, groups, member groups, actors, and member actors list the resources as directory entries. A space also has properties for roles, groups, and member groups. Discretionary access control (DAC) and role-based access control (RBAC) policies are defined in terms of the subjects from the directory (see Figures 3 and 13).

Membership: A membership in a community or space is represented by access control policies in the nested scopes of communities, spaces, and space items (space items are sub-containers in a space). The properties such as member groups and member actors are for indexing only while access control lists and roles define the operational aspects of membership. As a hypothetical example, let’s define a “viewer” membership in a project to mean that an individual can view the artifacts and post discussions to the forums in the project space, but may not update artifacts in the space. We further define “participant” membership for active participants who may update any artifact in the space. The UML collaboration diagram in Figure 14 shows how such membership constructs can be configured by access control model. The "participant" and "viewer" groups in conjunction with the access control policy serve as the template for this hypothetical membership construct.
Figure 14 A Hypothetical Membership Construct.

Once the membership scheme is defined, the users can be added to one of the membership groups. In the configuration in Figure 14, both userA and userB are members of the project space. UserA is a member of the group which has read/write access to the whole space and so can actively participate in the project space. UserB is a viewing member who has read/write access in the forum but only read access in the space. Thus userB is a viewer who may only post discussions in the forum. The "participants" and "viewers" groups can be created in the project space if they are used only for this project space.

The following code snippet describes how to create a community and add a user to the community. Role definitions are usually created in the community to provide uniform role assignments across the spaces of the community. The JPA framework can sort the entities in the transaction by the dependency of the entities to avoid referential constraint violations when it creates the entities one by one in the collaboration services.

The code examples also illustrate the cascade persist mechanism in action. We have declared JPA CascadeType.PERSIST annotation on the parent-to-child containment properties of Community, Space, and other parental classes. When a new community object is created under an organization, the community becomes persistent without requiring an explicit call to EntityManager.persist() method because the parent organization is persistent. Similarly when user, group, role, role definition, and space objects are created in the community, these objects become persistent. This is one of the features that makes JPA programming model seamless.

```java
SessionContext ctx;
Community organization;
Owner owner;
...

Date dt = new Date();
Community community = new Community(organization, dt);
community.setName("ICOM Community");
community.setOwner(owner);
community.setDescription("A developer community for ICOM");

User user = new User(community, dt);
user.setName("Eric");
user.setFamilyName("Chan");
```
```java
EntityAddress address1 = new EntityAddress();
URI emailAddress1 = new URI("mailto:" + "eric.s.chan" + "@" + "oracle.com");
address1.setAddress(emailAddress1);
address1.setAddressType("BUSINESS_1");
user.addAddress(address1);

EntityAddress address2 = new EntityAddress();
URI emailAddress2 = new URI("mailto:" + "eschan" + "@" + "acm.org");
address2.setAddress(emailAddress2);
address2.setAddressType("BUSINESS_2");
user.addAddress(address2);

EntityAddress address3 = new EntityAddress();
URI emailAddress3 = new URI("mailto:" + "eric.s.chan" + "@" + "oracle.com");
address3.setAddress(emailAddress3);
address3.setAddressType("BUSINESS_1");
user.setPrimaryAddress(address3);

RoleDefinition participantRoleDef = new RoleDefinition(community, dt);
participantRoleDef.setName("Participant");
participantRoleDef.addPrivilege(BeehivePrivilegeEnum.WORKSPACE_MGR);
participantRoleDef.addPrivilege(BeehivePrivilegeEnum.CONTENT_MGR);
participantRoleDef.addPrivilege(BeehivePrivilegeEnum.USER_MGR);
participantRoleDef.setOwner(owner);

RoleDefinition viewerRoleDef = new RoleDefinition(community, dt);
viewerRoleDef.setName("Viewer");
viewerRoleDef.addPrivilege(BeehivePrivilegeEnum.CONTENT_USER);
viewerRoleDef.addPrivilege(BeehivePrivilegeEnum.FORUM_WRITER);
viewerRoleDef.setOwner(owner);

try {
  ctx.getUserTransaction().commit();
} catch (Exception ex) {
  ...  
}
```

The following code snippet describes how to create a space with participant and viewer members. It looks up the role definitions in the parent community. When the transaction is committed, the JPA framework creates the objects in the collaboration services through the data access connector.
participants.setName("Participants");
participants.setDescription("Participants of community");
participants.addMemberActor(user);
participants.setOwner(user);

Role projectParticipant = new Role(icomJPAProjectSpace, participantRoleDef, dt);
projectParticipant.setName("ICOM JPA Project Participant");
projectParticipant.addMemberActor(participants);
projectParticipant.setAssignedScope(icomJPAProjectSpace);
projectParticipant.setOwner(user);

Group viewers = new Group(icomJPAProjectSpace, dt);
viewers.setName("Viewers");
viewers.setDescription("Viewers of project");
viewers.setOwner(user);
for (User aUser : subscribers) {
    viewers.addMemberActor(aUser);
}

EntityAddress address = new EntityAddress();
URI emailAddress = new URI("mailto:icomJpaDev@java.net");
address.setAddress(emailAddress);
address.setAddressType("BUSINESS_1");
viewers.setPrimaryAddress(address);

Role projectSubscriber = new Role(icomJPAProjectSpace, viewerRoleDef, dt);
projectSubscriber.setName("ICOM JPA Subscribers");
projectSubscriber.addMemberActor(viewers);
projectSubscriber.setAssignedScope(icomJPAProjectSpace);
projectSubscriber.setOwner(user);

try {
    ctx.getUserTransaction().commit();
} catch (Exception ex) { ...

Artifact Branch
The elements of the spaces are artifacts. Artifact includes generic components like Folder, Document, and Message. Heterogeneous folder is a common type of Folder. UnifiedMessage is a common type of Message. See Figure 15.
Artifact: An artifact is a result of a communication, cooperation, content creation, or in general a collaboration activity. Sending a message is an example of a collaboration activity that results in a message artifact. An artifact’s properties include description, user creation date, user last modification date, extensible properties, and viewer properties. See Figure 16. Unlike the read-only creation date and last modification date properties inherited from Entity, user creation date and user last modification date properties can be updated by applications. Artifact is relationship bondable so has a property for a set of relationships of an artifact.
Folder: A folder is a class of artifact to contain other artifacts. Every folder except the root folder has at least one parent folder. The parent of the root folder is a space. Subclasses of folder should enforce their own semantics on elements of the folder.

HeterogeneousFolder: A heterogeneous folder is an unconstrained folder to contain any type of artifacts. It is typically used for document library, inbox, and trash containers of a space. Heterogeneous folder has a property to hold elements, including documents and unified messages. See Figure 17.

![Figure 17 UML Class Diagram for HeterogeneousFolder.](image)

ICOM specification defines many types of folders which can be elements of the spaces. A space aggregates the collaboration activities across different folders, folders that more or less represent technology or protocol channels, to support the continuity of conversations, projects, tasks, and contexts. The specialized folders are defined as extension modules in the ICOM specification. The repertoire of specialized folders can potentially grow to include other advanced collaboration activities, such as decision support, simulation, command and control, business process monitoring, to name a few. Heterogeneous folders can be used to support inbox, outbox, document library, wiki pages, and trash folders in spaces, in addition to the following types of folders which are also commonly used for composing project workspaces (see Figure 18):

AddressBook: An address book is a folder that contains contacts, which can include bookmarks to addressable entities in the directories, addresses, and other personal entries.

Calendar: A calendar is a folder that contains time management artifacts such as occurrences and occurrence series.

TaskList: A task list is a folder that contains task management artifacts such as tasks and task assignments.

Blog: A blog (a blend for the term web log) is a folder that contains journal or log entries for access through web channel.
Forum: A forum is a folder that contains sub-forums, topics, announcements, and discussions.

Conference: A conference is a folder for visual, audio, and chat transcripts of the conference sessions. It also specifies the current status, conference settings, past sessions, active session, and activity logs.

Figure 18 A Workspace Composed of Several Types of Folders.

In addition to the specialized folders for coordination, communication, and content management, ICOM specification includes the following general categories of model:

- metadata (formal taxonomy, social linking/tagging folksonomy),
- process integration (workflow, policy, subscription, content change notification), and
- activity streams.

**Metadata Model**

ICOM defines the Relationship and Marker metadata model. Relationship can be associated with any relationship bondable entity. Almost all types of Entity, except Relationship type, are relationship
bondable. Therefore, a relationship cannot be relationship bonded by other relationships. Marker, which includes Category and Tag, can be associated with any type of Entity.

A relationship is an entity that relates a set of entities by a predicate. A relationship definition is an entity that defines the type of a relationship, including a name and a description of the relationship type, types of source entity and target entities of the relationship, and definition of properties in the relationship. See Figure 19.

Figure 19 UML Class Diagram for Relationship.

A marker is an artifact that groups together entities by a criterion. Markers can be flat or hierarchical. Flat markers are modeled by tag and hierarchical markers are modeled by category. See Figure 20. In some cases when a user applies a marker to an entity, the marker application should be private such that only the user who applies the marker can browse or locate the entity through the marker, especially when a marker created by a user is visible only to the user. A marker is listed in the markers property of one or more entities.

Figure 20 Marker Classes.
A tag is a marker that labels entities by a keyword. A tag has properties for application count that records how many times a tag has been applied on entities. Each application is represented by a tag application object. See Figure 21. A tag application has properties for attached entity, applied by actor, and application date. A tag application is listed in the tag applications property of an entity.

![Figure 21 UML Class Diagram for Tag.](image)

A category is a marker that classifies entities by taxonomy. A category has properties for super category and sub-categories. It also has property definitions that specify the name, type, cardinality, and facet of the properties for category applications. See Figure 22. A category application is an instance of association between a category and a specific entity. A category application has properties for category, attached entity, and extensible properties corresponding to the property definitions in the category. A category application is listed in the category applications property of an entity.

![Figure 22 UML Class Diagram for Category.](image)

The following code snippet shows how to create a hierarchy of categories. The root category represents “Problem Domain” which has “Performance Problem” and “Security Problem” as two sub-categories. The classified by property for each category application can be specified by a property definition. These categories can be used to classify the documents in a library.

```java
SessionContext ctx;
Space space;
...
Date dt = new Date();
Category rootCategory = new Category(space, dt, dt);
```
rootCategory.setName("Problem Domain");
rootCategory.setIsAbstract(true);

PropertyDefinition classifiedBy = new PropertyDefinition("Classified By");
classifiedBy.setDescription("The person who applies the category on the entity");
classifiedBy.setCardinality(Cardinality.Single);
classifiedBy.setPropertyType(BeehivePropertyTypeEnum.ID);
classifiedBy.setQueryable(true);

Category performanceProblem = new Category(rootCategory, dt, dt);
performanceProblem.setName("Performance Problem");
performanceProblem.setIsAbstract(false);

Category securityProblem = new Category(rootCategory, dt, dt);
securityProblem.setName("Security Problem");
securityProblem.setIsAbstract(false);

try {
    ctx.getUserTransaction().commit();
} catch (Exception ex) {
...
}

Supposing we have also constructed a category hierarchy for solutions taxonomy, the following code snippet shows how to relate the “Performance Problem” domain with the “Tuning Guide” and “Capacity Planning” solution domains using a relationship.

SessionContext ctx;
...

Date dt = new Date();
RelationshipDefinition problemSolution = new RelationshipDefinition(space, dt);
problemSolution.setName("Problem-Solution Association");
problemSolution.setDescription("Association from a problem domain to a solution domain");

String queryText1 = "select c from Category c where c.name = 'Performance Problem'";
Query query1 = EntityManager.createQuery(queryText1);
List<?> list1 = query1.getResultList();
Category performanceProblem = (Category) list1.get(0);

String queryText2 = "select c from Category c where c.name = 'Tuning Guide'";
Query query2 = EntityManager.createQuery(queryText2);
List<?> list2 = query2.getResultList();
Category tuningSolution = (Category) list2.get(0);

String queryText3 = "select c from Category c where c.name = 'Capacity Planning'";
Query query3 = EntityManager.createQuery(queryText3);
List<?> list3 = query3.getResultList();
Category capacityPlanning = (Category) list3.get(0);
...

Relationship guide = new Relationship(problemSolution, performanceProblem, dt);
guide.setName("solution guide");
guide.addTargetEntity(tuningSolution);
guide.addTargetEntity(capacityPlanning);

try {
    ctx.getUserTransaction().commit();
} catch (Exception ex) {
...
}
The ICOM applications can let the user navigate from a problem report document classified under the “Performance Problem” domain to the performance tuning guides classified under the “Tuning Guide” and “Capacity Planning” domains.

```
String queryText1 = "select rd from RelationshipDefinition rd where rd.name = 'Problem-Solution Association' ";
Query query1 = entityManager.createQuery(queryText1);
List<?> list1 = query1 resultList();
RelationshipDefinition problemSolution = (RelationshipDefinition) list1.get(0);

String queryText2 = "select c from Category c where c.name = 'Performance Problem' ";
Query query2 = entityManager.createQuery(queryText2);
List<?> list2 = query2.getResultList();
Category performanceProblem = (Category) list2.get(0);
```

```
Document doc;
...
```

```
Set<CategoryApplication> problemClassifications = doc.getCategoryApplications();
for (CategoryApplication catApp : problemClassifications) {
    Category cat = catApp.getCategory();
    if (cat == performanceProblem) {
        Set<Relationship> relations = cat.getRelationships();
        for (Relationship bond : relations) {
            if (bond.getRelationshipDefinition() == problemSolution) {
                Set<RelationshipBondable> targets = bond.getTargetEntities();
                for (RelationshipBondable target : targets) {
                    Category solution = (Category) target;
                    Collection<Entity> solutionGuides = solution.getMarkedEntities();
                }
            }
        }
    }
}
```

**Content Model**

The Content model in Figure 23 is a common component of Document model (Figure 24) and UnifiedMessage model (Figure 25). A content object represents a piece of data in a document or message. Content, multi-content, simple content, and online content form a composite design pattern. Among the properties of a content, a content id is a unique identifier for a part of content in multi-part contents; a media type is a two-part identifier for Internet file formats as defined in RFC 2046 and additional RFCs including RFC 3236, RFC 1847, etc; a content disposition is defined in RFC 2183 to specify a presentation style.

A multi-content object represents multiple parts of a message or document. It is a composite content that can contain a list of simple or composite contents. A simple content holds a single piece of data. Among the properties of SimpleContent, content encoding specifies RFC 2616 content encoding applied to a content; character encoding specifies RFC 2616 character set of a content (a missing value means that a content should be treated as binary or raw); content language specifies RFC 2616 content language for a content (a missing value means non-natural language content). An online content holds an online artifact attached to a document, message, or occurrence. An online artifact, such as a conference, chat room, or forum, must be rendered as a URL when a message or occurrence is delivered to external recipients.
Document Model

A document is a versionable artifact that can contain single or composite contents for any assortment of media types. See Figure 24. ICOM version control model follows the CMIS version control model specified in Section 2.1.9 of Content Management Interoperability Services Version 1.0[9].

ICOM extends the CMIS version control model with a concept called representative copy. A versionable artifact is a representative copy, specific versioned copy, or private working copy of an artifact version series. A container of a versionable artifact can contain a representative copy of a version series so that it provides an artifact view of the latest state of the version series. Starting from a representative copy in a container, an actor can traverse a version series to retrieve any versioned copy or private working copy.

When a versionable artifact is not under version control, a representative copy of a versionable artifact is the only version of a version series and represents the versionable artifact itself, i.e. there is only one object identifier so far. When a versionable artifact is placed under version control:

- A representative copy of a versionable artifact is a versionable artifact which has its own object identifier that is different from the object identifier of any versioned copy or private working copy of the versionable artifact. It retains the object identifier it has when the artifact is created. Its version type changes from RepresentativeCopy to ViewOnlyRepresentativeCopy.
• A representative copy of a versionable artifact provides a view of a version series, depending on the check out state of the version series and the user loading the artifact. If the current user loading a representative copy is the same user who checks out from a version series, the representative copy is a copy of the content and state of a private working copy. Otherwise, the representative copy is a copy of the content and state of the latest versioned copy in a version series.

A specific versioned copy of a versionable artifact is a “deep” copy of the content and state of a versionable artifact, preserving its content and state at a certain point in time. Each versioned copy of a versionable artifact is itself a versionable artifact, i.e. it has its own object identifier. A versioned copy has a version number, label, and check in comment.

A private working copy of a versionable artifact is a versionable artifact created by a checkout operation on a versionable artifact under version control. The properties for a private working copy can be identical to the properties of a versioned copy of a versionable artifact on which a checkout operation is performed. Its object identifier must be different from that of the representative copy or any versioned copy. A private working copy can be saved to the version series for sharing and co-editing, however, it needs not be visible to users who may only have permissions to view other versioned copies in a version series.
The following code snippet shows how to create a new version series by creating a document and checking out the document. It shows that when a document is created, it becomes a representative copy. After the document is checked out to form a new version, the original document remains as the representative copy of the version series. The document folder continues to contain the representative copy of the document. The private working copy from the check out operation is a new artifact with a new object identifier.

```java
public Version createDocumentVersion(SessionContext ctx, icom.Service service, HeterogeneousFolder documentFolder, String documentName, File file) {
    Date dt = new Date();
    Document representativeCopy = new Document(documentFolder, dt, dt);
    representativeCopy.setName(documentName);
    representativeCopy.setDescription("A document created from ICOM JPA");
    representativeCopy.setUserLastModificationDate(dt);
    SimpleContent content = new SimpleContent();
    content.setDataFile(file);
    content.setContentType("text/html");
    content.setLanguage(Locale.getDefault());
    content.setCharacterEncoding("UTF-8");
    content.setContentEncoding("8bit");
    representativeCopy.setContent(content);
    try {
```

Figure 24 UML Class Diagram for Document.
ctx.getUserTransaction().commit();
} catch (Exception ex) {
    ...
}

Document privateWorkingCopy = (Document) service.checkout(representativeCopy, "starting a new version series");
VersionControlMetadata metadata = privateWorkingCopy.getVersionControlMetadata();
Version v = (Version) metadata;
return v;
}

The following code snippet shows how to create a new version of a document from a private working copy. It shows that the state of the private working copy must be committed to the repository before calling the check in operation.

public Version addVersion(SessionContext ctx, icom.Service service, Document privateWorkingCopy) {
    privateWorkingCopy.setDescription("A new version of document created from ICOM JPA");
    Version version = (Version) privateWorkingCopy.getVersionControlMetadata();
    version.setName("Second version");
    Date dt = new Date();
    version.setCheckinComment("A check in comment");
    try {
        ctx.getUserTransaction().commit(); // upload the private working copy
    } catch (Exception ex) {
        ...
    }
    Document versionedCopy = (Document) service.checkin(version);
    return version;
}

The following code snippet shows how to navigate from a document to the version control metadata.

VersionControlMetadata metadata = doc.getVersionControlMetadata();
Versionable rep = metadata.getRepresentativeCopy();
if (metadata instanceof VersionSeries) {
    VersionSeries series = (VersionSeries) metadata;
    Date dt = series.getVersionSeriesCheckedOutOn();
    Actor actor = series.getVersionSeriesCheckedOutBy();
    String comment = series.getVersionSeriesCheckoutComment();
    Version version = series.getLatestVersion();
} else {
    Version version = (Version) metadata;
    String name = version.getName();
    String label = version.getVersionLabel();
    Integer number = version.getVersionNumber();
}

Unified Message Model

A message is a unit of conversation that holds a simple content or multipart message contents in a content property. It has a single sender. The delivered time is the time when a message is delivered to a given recipient and the sent time of a message is represented by a user creation date and time of the message. The name property holds the subject of a message.

A unified message (see Figure 25) is a special type of message delivered electronically over a computer, voice, fax, and other networks. A unified message can be one of these types:

- Email is a type of message that is delivered electronically over a computer network.
• Voice is a type of message that contains a voice or audio stream.
• Fax is a type of message that contains an image transmitted via phone lines using the fax protocol.

Figure 25 UML Class Diagram for UnifiedMessage.

ICOM unified message supports a persistent object identifier. This differentiates ICOM from SMTP, IMAP, POP protocols and Java Mail API. RFC 2822 Internet Message Format specifies a MIME header “Message-ID” to identify the messages. The uniqueness of the message id should be guaranteed either by the client or server which generates it. IMAP on the other hand assigns a 32-bit UID to messages in the ascending order as they are added to a container. ICOM object identifier supersedes both MIME message id and IMAP UID. The MIME message id is preserved as part of MIME headers in the unified message. The IMAP UID can be associated with a unified message, and for that matter any MIME convertible artifact, using an extensible property on Artifact (Figure 16). RFC 2822 also specifies “References” and “In-Reply-To” MIME header attributes that use message id to represent the message threads. These two attributes are also preserved as part of MIME headers in the unified message. ICOM JPA framework allows the applications to create the reference and in-reply-to chains using the
Relationship objects (see Figure 19), by crawling the messages as they are added to a workspace and correlating the “Message-ID,” “References,” and “In-Reply-To” attributes in the MIME headers. It is possible to find multiple unified messages with the same message id, for a legitimate reason when a message is sent to multiple recipients who then move or copy their messages into the same container. It can also result from a rogue client forging a message with the message id of another message. ICOM object identifier can be useful in resolving these conflicts.

ICOM definesUnifiedMessage and Document as MimeConvertible. This design represents another substantive departure from other API’s such as Java Mail API. When a MimeConvertible artifact from a folder is attached to a message, the artifact contents are copied as an attachment into the message. The attached object in a message is semantically compatible with an embedded object in JPA, i.e. the attached object does not have object identifier, change token, and is not sharable among entities (nor among more than one attribute or level of MIME hierarchy in the containing message entity). In JPA specification, Entity and Embeddable categories are disjoint. Since UnifiedMessage and Document are annotated as JPA Entity, these classes cannot be used to represent embedded objects. In order for the ICOM POJO classes to be portable to standard compliant JPA providers, we need to introduce embeddable UnifiedMessageAttachment and DocumentAttachment classes for JPA binding.

It is a common scenario for ICOM applications to let a user forward email messages from her INBOX as attachments in a new email message. The following code snippet shows how the application can take a collection of UnifiedMessage entities, clone each entity, and add the clones as embedded objects in the MultiContent. The UnifiedMessage entities being cloned can contain nested structures of embedded UnifiedMessageAttachment objects (representing long message threads).

```java
public UnifiedMessage forwardMessages(HeterogeneousFolder draftFolder, Collection<UnifiedMessage> attachments) {
    Date dt = new Date();
    UnifiedMessage newMsg = new UnifiedMessage(draftFolder, dt, dt);
    MultiContent forwards = new MultiContent();
    forwards.setMediaType(MultiContentType.Mixed);
    for (UnifiedMessage message : attachments) {
        message.getContent(); // make sure to load the content before cloning (analogous to detaching)
        UnifiedMessageAttachment msg = message.cloneEmbeddable(); // clone the message
        forwards.addPart(msg); // the clone is attached or embedded as a MIME part
        message.addFlag(UnifiedMessageFlag.Forwarded); // the original message is flagged as “Forwarded"
    }
    multiContent.addPart(forwards);
    newMsg.setContent(multiContent);
    ...
}
```

The following code snippet shows an application saving the document attachments in a message to a folder. This example also involves cloning an embedded document object and saving the clone as an artifact in a document folder.

```java
MultiContent content = message.getContent();
if (content instanceof MultiContent) {
    for (MimeConvertible mime : content.getParts()) {
        if (mime instanceof DocumentAttachment) {
            Document doc = document.cloneEntity();
            doc.setParent(documentFolder);
            entityManager.persist(doc); // this explicit persist operation is not needed with use of Cascade Persist
These scenarios may represent the drag and drop activities in a user interface. Dragging and dropping a message or document from a folder into a message or from a message into a folder is a copy and paste (or clone) operation. The above examples involve cloneEmbeddable and cloneEntity methods, which are essentially clone operations. The cloneEmbeddable method shall clone a UnifiedMessageAttachment or DocumentAttachment, respectively, from UnifiedMessage or Document entities. Likewise, the cloneEntity method shall clone UnifiedMessage or Document entities, respectively, from UnifiedMessageAttachment or DocumentAttachment.

ICOM JPA framework can support improved fidelity for transporting artifact attachments by propagating the properties such as name, description, created by, creation date, last modified by, last modification date, user creation date, user last modification date, etc., defined on Entity and Artifact as MIME headers. As is normally the case when copying artifacts within folders, an artifact’s properties for parent, owner, attached markers, category applications, tag applications, relationships, and access control list are not copied.

We introduced a method isEditable() in POJO implementation of ICOM Entity to represent that some ICOM entities are immutable under certain conditions. In particular, a versioned copy of a document (when versionType is VersionType.VersionedCopy or VersionType.ViewOnlyRepresentativeCopy) and a delivered unified message (when mode is UnifiedMessageEditMode.DeliveredCopy) are immutable. The ICOM JPA framework throws an uncheck exception when an application updates the state of an immutable entity. To provide the same functionality when the POJO classes are ported to a standard JPA provider, we need to deploy a JPA entity lifecycle callback listener for PreUpdate callbacks that will throw an unchecked/runtime exception when an application updates an immutable entity. If the immutable entity update exception is thrown while a transaction is active in the persistent context, the transaction will be marked for rollback.

**Conclusion**

This is part 1 of the three parts series introducing the java.net incubation project[14] for JPA ICOM framework. Part 1 presents the high-level concepts of ICOM, including directory, space, access control, metadata, content management, and unified message model, with programming examples for the ICOM JPA framework. Part 2 of this series will describe the details of ICOM extension modules, including Forum, Wiki, Calendar, TaskList, AddressBook, Conference, Workflow, Policy, Subscription, and Activity Stream, to name some. Part 3 will describe the design of the ICOM JPA framework.

The JPA programming model embodies design patterns that are well-suited for managing integrated collaboration object model (ICOM). ICOM removes the walls between the collaboration tools and exposes the data from behind the wall of applications. The ICOM JPA framework lowers the barrier for application developers to create common collaboration clients to interoperate with integrated collaboration platforms and standalone collaboration services across enterprise boundaries. The ICOM
JPA framework is an important initiative for the java.net community, ready to reap benefits from community contributions.

**Acknowledgements**

I am thankful to the OASIS ICOM TC members for their inputs to the ICOM specification. Many ideas in this article originated from stimulating discussions with Stefan Decker, Deirdre Lee, Laura Dragan, Patrick Durusau, Peter Yim, Anthony Ye, Marc Pallot, Peter Saint-Andre, and Chancellor Pascale. I will like to thank Mike Keith for the technical review and constructive feedbacks that led to important changes in the article. Many thanks to Martin Chapman for his advice to me on the OASIS standard processes and a legal framework for incubating the ICOM JPA source code.

**References**


