Catalog Entities 1.1 Component Specification

1. Design

The Catalog database stores information about TopCoder components and applications.

This component defines entities to represent database information, providing also the O/R mapping for Hibernate.

A class diagram (Entities.zuml), a DB diagram (DataModel.png) and DDL for creating the DB tables (Catalog.ddl) are provided together with this specification.

1.1 Design Patterns

None.

1.2 Industry Standards

XML

JPA

Hibernate

JBoss

Informix Database

1.3 Required Algorithms

1.3.1Named queries

We must define three named queries in the Hibernae XML file in this way:

```
<query name="getAllPhases">
 SELECT p FROM Phase p
</query>
<query name="getActiveTechnologies">
  SELECT t FROM Technology t WHERE t.status=1
</query>
<query name="getActiveCategories">
  SELECT c FROM Category c WHERE c.status=1 and c.viewable=true
</query>
```

1.3.2O/R mappings

We must define the mappings for hibernate in an xml file. Where the entity has an id, we must define the sequence name for the id generator.

Here there is an example for the Phase entity

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="com.topcoder.catalog.entity">
    .....
      <class name="Phase" table="phase">
            <id name="id" column="phase_id">
            <generator
```

```
      class="com.topcoder.catalog.entity.IdGenerator">
                        <param name="sequence_name">PHASE_SEQ</param>
                  </generator>
                  </id>
                  <property name="description" column="description"
access="field"/>
            </class>
          ....
      </hibernate-mapping>
```

Use the DataModel provided with the distribution to find how to map objects.

1.4 Component Class Overview

Component:

This class represents a component that can actually be an application, assembly, testing or component.

The component can have many versions, and the current version can be retrieved from comp_versions table matching current_version with version field, and both component_id's fields.

Other versions will be stored in versions attribute.

It's important to understand that the current version is not necessarily the most recent row, since except for the first version, it points to a completed version. For example, when a new component is created, it will use version 1, so comp_versions.version and comp_catalog.current_version will be set to 1. Then, after the component is completed, another version is released. A new row will be created in comp_version with version 2, but comp_catalog.current_version will still be 1. When the new version is completed, this field will be changed to 2, pointing to that version.

Also notice that the version field is just an "internal" field that will start in 1 and increase by 1 each time. The user will see comp_versions.version_text instead, where version numbers like "1.1" can be stored.

Each component has a root category, stored in field root_category_id.

The component has as well a collection of categories (from 0 to n). Each category should have as its root ancestor the root category.

There can be some users (other than admins) authorized to view and change component data.

The users attribute provides a list of CompUser entities with the ids of the authorized users.

Also, there can be some clients authorized to change component data, represented by clients list.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

Category :

Categories are used to classify components. They can be related to other categories through parent_category_id field, creating a hierarchy.

On the top level, there are categories like "Java", "C++", ".Net", etc, whose parent_category_id is set to null. Then, other categories, like "JSF", "Swing", "Communication" are child of "Java".

Currently, there are about 60 categories.

Field status indicates whether the category is active or deleted.

Field viewable indicates whether the category should be displayed to users in order to be selected or not.

Root categories can be associated with a catalog (catalog table, containing rows for “Java”, “.Net”, etc). The name of the catalog must be retrieved in catalogName attribute in Catalog class.

In order to do that, the table navigation must be done through category_catalog.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

CompUser :

This class represents an association between a component and a user, meaning that the user can view

and change component data.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

CompClient :

This class represents an association between a component and a client.

This entity also includes a list of the users for the client in the users attribute, retrieved via user_client table. This list is read only.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

Status :

This class represents an enumeration of the possible status. Each status has a different status id.

CompVersionDates :

This class contains dates for each phase of a component version, as well as comments for those dates and some additional information.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

CompVersion :

This class represents a component version.

When a new version is created, version field must be increased by one, while version_text field will be entered by the user.

The version points to the current phase using phase_id to look up in phase table.

Fields phase_time and price represent the start date and price of the current phase. They are now redundant, since table comp_version_dates provides that information with more detail.

For each phase of the version, there could be a row in comp_version_dates providing different dates for the version, as well as comments and the price. The field versionDates uses a map whose key is the phase_id and the value is a VersionDates entity to represent this.

The version contains a list of technologies in technologies attribute. This list must be retrieved navigating through comp_technology table.

The component version can have a forum associated. This is represented in the entity by forum attribute.

The component version can have a link (currently an svn link where component files are stored), represented by link attribute.

The component version can have a list of documents accosicated to it. This list must be retieved through comp_documentation table.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

Technology :

This class represents a technology that a component version uses, like “XML”, “EJB”, “Spring” and so on.

The status is used to indicate whether the technology is active or it was logically deleted.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

Phase :

This class represents a component phase, like collaboration, design, development or completed.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

CompForum :

This class stores the forum associated with the component version.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

CompLink :

This class stores a link for the component version; currently it’s an SVN link for the component files.

This class implements the Serializable interface.

Validation of parameters in setters is outside the scope of the Catalog Entities component.

IdGenerator :

This class generates the id for the entities when they are stored in persistence. This is achieved through xml hibernate mapping.

StatusUserType :

This class used by Hibernate to store/retrieve/update fields of type **Status**, and to use it in 'where' clauses.

CompDocumentation:

This class stores the documentation associated with the component compVersion.

This class is mutable and not thread safe.

This class implements the Serializable interface.

Validation of parameters is not performed in this class. It's supposed to be a caller's responsibility.

1.5 Component Exception Definitions

HibernateException:

This exception is thrown by IdGenerator whenever an error occurs while generating an id.

1.6 Thread Safety

This component is not thread safe, because all classes (except Status ans StatusUserType) have changeable fields. It can be used in a thread safe manner if deployed in Hibernate. However, avoiding reading and writing on different threads simultaneously is needed for complete thread safety.

2. Environment Requirements

2.1 Environment

- At minimum, Java1.5 is required for compilation and executing test cases.

2.2 TopCoder Software Components

- Id Generator 3.0: used for id generation.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.catalog.entity

3.2 Configuration Parameters

None.

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Follow demo.

4.3 Demo

To interact with persistence, you need an EntityManager.

To configure EntityManagerFactory you can use persistence.xml file, like the following:

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

             version="1.0"

             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

    <persistence-unit name="catalog_manager">

        <provider>org.hibernate.ejb.HibernatePersistence</provider>

        <properties>

            <property name="hibernate.dialect" value="org.hibernate.dialect.InformixDialect"/>

            <property name="hibernate.show_sql" value="true"/>

            <property name="hibernate.connection.driver_class"
value="com.informix.jdbc.IfxDriver"/>

            <property name="hibernate.connection.username" value="informix"/>

            <property name="hibernate.connection.password" value="123456"/>

            <property name="hibernate.connection.url"
value="jdbc:informix-sqli://192.168.131.66:9088/sysuser:INFORMIXSERVER=server_name"/>

        </properties>

    </persistence-unit>

</persistence>
```

Persisting entities.

```
Phase phase = new Phase();
phase.setDescription(“screening”);
entityManager.persist(phase);
```

Finding entities.

```
Phase phase = entityManager.find(Phase.class, phaseId);
```

Create queries.

```
Query query = entityManager.createQuery("select p from Phase p where
p.id=:id");
query.setParameter("id", phaseId);
Phase found = (Phase) query.getSingleResult();
```

Obtaining named queries.

```
query = entityManager.createNamedQuery("getAllPhases");
@SuppressWarnings("unchecked")
final List<Phase> phases = query.getResultList();
```

Removing phases.

```
entityManager.remove(entityManager.getReference(Phase.class,
phaseId));
```

Creating components and component versions.

```
// create component
final Component component = new Component();
component.setDescription("Component description");
component.setFunctionalDesc("Functional description");
component.setName("Component name");
component.setShortDesc("Short Desc.");
component.setStatus(Status.NEW_POST);
// create version
final CompVersion version = new CompVersion();
version.setComments("Version comments");
version.setPhasePrice(500);
version.setPhaseTime(parseDate("2007/12/21"));
version.setSuspended(false);
version.setVersion(1L);
version.setVersionText("1.0");
// create forum
final CompForum compForum = new CompForum();
compForum.setVersion(version);
version.setForum(compForum); // assign to the version
```

```
// create link
final CompLink compLink = new CompLink();
compLink.setLink("some svnlink");
compLink.setVersion(version);
version.setLink(compLink); // assign to the version
// create documentation
 // note: newly added in version 1.1
 final CompDocumentation compDocumentation = new CompDocumentation();
 compDocumentation.setDocumentName("my doc");
 compDocumentation.setDocumentTypeId(300L);
 compDocumentation.setUrl("software.topcoder.com");
 compDocumentation.setCompVersion(version);
List<CompDocumentation> documentation = new
ArrayList<CompDocumentation>();
 documentation.add(compDocumentation);
version.setDocumentation(documentation); // assign to the version
// assign phase, which is already in the database
version.setPhase(getEntityManager().find(Phase.class, 1L));

// populate version phase dates
final Map<Long, CompVersionDates> dates =
populateVersionDates(version);

version.setVersionDates(dates);
version.setComponent(component);

// set the current version
component.setCurrentVersion(version);

// set the root category (categories are in the database
component.setRootCategory(getEntityManager().find(Category.class,
2L));

// assign categories which this component belongs to
component.setCategories(Arrays.asList(
   getEntityManager().find(Category.class, 2L),
   getEntityManager().find(Category.class, 3L)));
```

```
// assign component clients
final Set<CompClient> clients = populateClients(component);
component.setClients(clients);

// assign component users
final Set<CompUser> users = populateUsers(component);
component.setUsers(users);

// start the transation
entityTransaction.begin();
// persist the version first
entityManager.persist(version);
// persist the component, this will update version as well to bind it
to the component
entityManager.persist(component);
// commit the transation
entityTransaction.commit();
```

Removing components

```
entityTransaction.begin(); // start transaction
version.setComponent(null); // clear reference to component
component.setVersions(null); // remove to versions associations
component.setUsers(null); // remove to users associations
component.setCategories(null); // remove to categories associations
component.setClients(null); // remove to clients associations
entityManager.merge(version); // update version without
entityManager.remove(entityManager.getReference(Component.class,
component.getId())); // remove component
entityManager.remove(entityManager.getReference(CompVersion.class,
version.getId())); // remove version
entityTransaction.commit(); // commit finally
```

5. Future Enhancements

None at the moment.