[TopCoder]

Catalog Entities 1.0 Requirements Specification

1. Scope

1.1 Overview

The Catalog database stores information about TopCoder components and applications.

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

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.

1.2 Logic Requirements

1.2.1 Entities and mapping

All the entities in the diagram must be implemented. Fields, methods and constructors can be added if needed. If you consider that changes are needed to the provided interface, please ask in the forums first.

Notice that `create_time` and `modify_date` fields present in several tables should not be mapped to entities, and they are automatically filled by the database.

The tables where records can be inserted must generate its unique key using Id Generator component.

The following tables must be read only:

- categories
- category_catalog
- catalog
- user_client
- technology_types
- phase

The mapping must be done in one or more XML files using JPA with Hibernate extensions.

1.2.1.1 Component (comp_catalog table)

It 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.

1.2.1.2 Category (categories table)

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_id` indicates whether the category is active (1) or deleted (0).

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`.

1.2.1.3 CompVersion (comp_versions table)

It 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.

1.2.1.4 Phase (phase table)

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

1.2.1.5 Technology (technology_types table)

It 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.

1.2.1.6 CompVersionDates (comp_version_dates table)

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

1.2.1.7 CompForum (comp_jive_category_xref table)

It stores the forum associated with the component version.

1.2.1.8 CompLink (comp_link)

It stores a link for the component version; currently it’s an svn link for the component files.

1.2.1.9 CompUser (comp_user table)

It represents an association between a component and a user, meaning that the user can view and change component data.

1.2.1.10 CompClient (comp_client table)

It 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.

1.2.2 Named Queries

The following named queries must be defined:

1.2.2.1 Named Query `getActiveCategories`

Return a list of all the categories having status_id = 1 and viewable=1

1.2.2.2 Named Query `getActiveTechnologies`

Return a list of all the technologies having status_id = 1.

1.2.2.3 Named Query `getAllPhases`

Return a list of all the phases.

1.2.3 Auditing

The component must audit creation, update and deletion of entities using Auditor 2.0 component. Each audit will contain information about who performed it, when it was performed and the information that was modified (before and after values).

The auditing data will occur in their own separate tables, and those tables will be read only from within the application.

The designer must provide the DDL for the auditing tables.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

Other components will rely on the entities provided by this component to provide higher level services concerning catalog database.

1.5 Future Component Direction

None.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

See file Entities.zuml

2.1.3 Environment Requirements

- Development language: Java 5.0
- Compile target: Java 5.0

2.1.4 Package Structure

com.topcoder.catalog.entity

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

none

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

- JPA (with Hibernate extensions)

3.2.2 TopCoder Software Component Dependencies:

- Id Generator 3.0
- Auditor 2.0

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

Informix Database

JBoss

Java Persistence API (JPA)

Hibernate

3.2.4 QA Environment:

- RedHat Enterprise Linux 4
- JBoss 4.2 GA

- Java 1.5
- Informix 10.00.UC 5

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.