Client Project Entities DAO 1.2 Component Specification

Additions introduced in v1.2 are in red.
Changes/Fixes introduced in v1.2 are in blue.
Changes/Additions introduced from previous upgrades are in purple.

1. Design

This component provides all necessary entities and DAO interfaces and implementations for the Client and Project Entities architecture. This component defines a number of entity classes, along with the necessary annotations for those entities to be used with Hibernate JPA extensions. In addition, DAO interfaces are provided and basic EJB3 implementations are be part of the design. The basic architecture of the DAO pieces is modeled on the CaveatEmptor example provided here:

http://www.hibernate.org/400.
This component will be used to represent the clients and projects for TopCoder customers.

Changes in version 1.1:
ProjectDAO is updated, with the following methods added:

- **addUserToBillingProjects**: Adds the user with given user name to the billing projects that are specified with project IDs.
- **removeUserFromBillingProjects**: Removes the user with given user name from the billing projects that are specified with project IDs.
- **getProjectsByClientId** :Get all projects associated with the given client id.

In version 1.2, a new budget attribute is added into Project entity, two new methods are added into ProjectDAO, one for updating projects' budget, the other for getting all users' names who have access of the billing project.

1.1 Design Patterns

DAO Pattern – this component provided a series of DAO interfaces and implementations, which follows this pattern.

Strategy pattern – the bean classes created in this design use the EntityManager injected in a strategic manner.

DTO pattern – all the entities from this design are Data Transfer Objects.

Version 1.2 doesn't bring any new design patterns.

1.2 Industry Standards

- EJB 3.0
- JPA 1.0

1.3 Required Algorithms

There are no complicated algorithms in this design.

Version 1.2 doesn't bring any new complex algorithms and the method documents in TCUML are enough for development.

The sections below are just design considerations used in this design.

1.3.1 Entities and Hibernate

All entities are just basic data holders that represent the entities used by the framework. They all implement the Serializable interface. Also, each entity is serializable to XML, for use with the Client and Project Services component. This should is done through the use of class annotations, like:

@XmlAccessorType(XmlAccessType.FIELD), @XmlType(name = "..." propOrder = { "..." })

In addition to the XML serialization annotations mentioned above, each entity also contain the necessary annotations to be used with the Hibernate EntityManager, outlined here:

http://www.hibernate.org/397.html
The provided database schema is used to generate the necessary table and column information.

1.3.2 DAO interfaces and implementations

The DAO interfaces provided the required business operations.

All DAO interfaces are extended by Local and Remote interfaces.

Implementations of DAO interfaces, of local and remote interfaces are stateless session beans.

1.3.3 DAO searching

TopCoder Search Builder 1.4 component is used to perform search of the Entities using a given Filter.

Search Builder 1.4 supports generating HQL using a provided Filter instance, so this component is be used to perform searching against the Hibernate data store.

Below is the ***public List<T> search(Filter filter)*** method:

TopCoder component Search Builder 1.4 should be properly configured before usage:
Please see TopCoder component Search Builder 1.4 how to create the configuration needed to use that component (it's Demo and CS are quite straight forward).
----------Conventions over the Search Builder configuration----------
SearchBundles should be configured with HibernateSearchStrategies.
For each Entity a separate HibernateSearchStrategy should be used (configured with aliases and searchable fields with values exactly like the properties of the entities), and property name for HibernateSearchStateges in the configuration should be: HibernateSearchBundle_ENTITY_NAME
(sample: HibernateSearchBundle_Project).

```
EqualToFilter equalToFilter = new EqualToFilter("deleted", new Boolean(false));
AndFilter resultedFilter = new AndFilter(filter, equalToFilter);
List<T> entities = searchFilterUtility.search(resultedFilter);
return entities;
```

1.3.4 Entity Deletion

As part of the Client and Project Entities application requirements, entities are not permanently deleted from the database. Because of this, each entity has an "isDeleted" flag that is set to "true" when the entity is deleted by calling DAO.delete(entity). When retrieving information, all rows where is_deleted is true are ignored. When deleting, the entity are updated with that flag set to true, and then saved to the database.

Below is the ***public void delete(T entity)*** method:

```
Id id = entity.getId();
```

```
T persitedEntity = retrieveById(id);
persitedEntity.setDeleted(true);
entityManager.merge(persitedEntity);
```

Below is the ***public List<T> searchByName(String name)*** method to show the requirement regarding retrieval of entities that are marked as deleted:

```
String queryString = "select e from " + entityBeanType.getCanonicalName() +" e"
+ "where e.name = :name"
+ " and e.deleted = false";
Query query = entityManager.createQuery(queryString);
query.setParameter("name", name);

// Involves an unchecked conversion:
List<T> entities = query.getResultList();

return entities;
```

1.3.5 Dependent Updates

When updating Projects, the contained children do not have to be part of the update, only the Project instance provided. It is expected the user will make separate calls to update the children. This same practice applies to all entities with complex dependent fields.

This requirement of updates is handled by annotations on entities:

```
@JoinColumn(… updatable = false,
        insertable = false)
```

Below is the ***public T save(T entity)*** method:

```
return entityManager.merge(entity);
```

1.3.6 Transactions

All access to the database that updates creates, or retrieves from multiple tables happen in a transaction. If an error occurs, the transaction is rolled back gracefully.
Exceptions are configured to handle this:
@**ApplicationException(rollback=true)** - to notify the EJB container to roll back the transaction.

1.4 Component Class Overview

1.4.1 com.topcoder.clients.model

AuditableEntity <<abstract, class, @MappedSuperclass>>
Usage:
This class represents the AuditableEntity java bean. An AuditableEntity can contain a id, createUsername, createDate, modifyUsername, modifyDate, name, deleted.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method).
This entity corresponds to a mapped super class.
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (implements Serializable interface).
Annotations:
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "auditableEntity",
propOrder = {"id", "createUsername", "createDate", "modifyUsername","modifyDate", "name", "deleted"})
@MappedSuperclass
Implementation notes:

implements Serializable;
Thread safety:
This class contains only mutable fields so therefore it is not thread safe.

Project <<class, @Entity>>
Usage:
This class represents the Project java bean. A Project can contain a company, active, salesTax, pOBoxNumber, paymentTermsId, description, projectStatus, client, childProjects, parentProjectId, manualPrizeSetting, budget.
See base class for other available properties.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method).
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (base class is Serializable).
Annotations:
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "project",
propOrder = {"company", "active", "salesTax", "pOBoxNumber", "paymentTermsId", "description","projectStatus", "client", "childProjects", "parentProjectId", "manualPrizeSetting", "budget"})
@Entity - this annotation should be used to mark this java bean as an entity and to ease its use with JPA.
@Table(name = "project")
Implementation notes:
extends AuditableEntity;
Thread safety:
This class contains only mutable fields so therefore it is not thread safe.

Company <<class, @Entity>>
Usage:
This class represents the Company java bean. A Company can contain a passcode.
See base class for other available properties.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method).
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (base class is Serializable).
Annotations:
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "company",
propOrder = {"passcode"})
@Entity - this annotation should be used to mark this java bean as an entity and to ease its use with JPA.
@Table(name = "company")
Implementation notes:
extends AuditableEntity;
Thread safety:
This class contains only mutable fields so therefore it is not thread safe.

ClientStatus <<class, @Entity>>
Usage:
This class represents the ClientStatus java bean. A ClientStatus can contain a description.
See base class for other available properties.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method).
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.

This class is Serializable (base class is Serializable).
Annotations:
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "clientStatus",
propOrder = {"description"})
@Entity - this annotation should be used to mark this java bean as an entity and to ease its use with JPA.
@Table(name = "client_status")
Implementation notes:
extends AuditableEntity;
Thread safety:
This class contains only mutable fields so therefore it is not thread safe.

Client <<class, @Entity>>
Usage:
This class represents the Client java bean. A Client can contain a company, paymentTermsId, clientStatus, salesTax, startDate, endDate, codeName.
See base class for other available properties.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method).
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (base class is Serializable).
Annotations:
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "client",
propOrder = {"company", "paymentTermsId", "clientStatus", "salesTax","startDate", "endDate", "codeName"})
@Entity - this annotation should be used to mark this java bean as an entity and to ease its use with JPA.
@Table(name = "client")
Implementation notes:
extends AuditableEntity;
Thread safety:
This class contains only mutable fields so therefore it is not thread safe.

ProjectStatus <<class, @Entity>>
Usage:
This class represents the ProjectStatus java bean. A ProjectStatus can contain a description.
See base class for other available properties.
This is a simple java bean (with a default no-arg constructor and for each property, a corresponding getter/setter method).
Any attribute in this bean is OPTIONAL so NO VALIDATION IS PERFORMED here.
This class is Serializable (base class is Serializable).
Annotations:
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "projectStatus",
propOrder = {"description"})
@Entity - this annotation should be used to mark this java bean as an entity and to ease its use with JPA.
@Table(name = "project_status")
Implementation notes:
extends AuditableEntity;
Thread safety:
This class contains only mutable fields so therefore it is not thread safe.

ProjectContestFeeAudit <<class, @MappedSuperclass >>
This class represents the audit fields for ProjectContestFee.

Thread Safety:
This class contains only mutable fields so therefore it is not thread safe.

Annotations:
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "projectContestFeeAudit", propOrder = { "createUsername","createDate", "modifyUsername", "modifyDate", "name", "deleted" })
@MappedSuperclass

Since:
Configurable Contest Fees v1.0 Assembly

ProjectContestFee <<class, @Entity >>
This class represents project contest fee.

Thread Safety:
This class contains only mutable fields so
therefore it is not thread safe.

Annotations:
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "projectContestFee")
@Entity
@Table(name = "project_contest_fee")

Since:
Configurable Contest Fees v1.0 Assembly

1.4.2 com.topcoder.clients.dao

GenericDAO<T extends AuditableEntity, Id extends Serializable> <<interface>>
Usage:
This interface represents the base GenericDAO business interface.
This interface defines the generic methods available for all the DAOs: retrieve entity by id, retrieve all entities, search entities by name, search entities using the given filter, save an given entity, delete an given entity.
Thread safety:
Implementations of this interface should be thread safe.

ProjectDAO<Project, Long> <<interface>>
Usage:
This interface represents the ProjectDAO business interface.
This interface defines the specific methods available for the ProjectDAO business interface: retrieve project by id and retrieve all projects.
See base class for other available operations.
Implementation notes:
extends GenericDAO;
Thread safety:
Implementations of this interface should be thread safe.

Changes in Configurable Contest Fees v1.0 Assembly, the following methods are added:
- searchProjectsByClientName : Search projects by client name

- searchProjectsByProjectName : Search projects by project name.
- saveContestFees : Save contest fees.
- getContestFeesByProject : get contest fees by project.

Changes in v1.1:
The following methods are added:
- addUserToBillingProjects : Adds the user with given user name to the billing projects that are specified with project IDs.
- removeUserFromBillingProjects : Removes the user with given user name from the billing projects that are specified with project IDs.
- getProjectsByClientId :Get all projects associated with the given client id.

In version 1.2, this interface adds two new method, updateProjectBudget updates the project's budget, getUsersByProject gets all users' names who have access of the billing project.

CompanyDAO<Company, Long> <<interface>>
Usage:
This interface represents the CompanyDAO business interface.
This interface defines the specific methods available for the CompanyDAO business interface: retrieve clients for company and retrieve projects for company.
See base class for other available operations.
Implementation notes:
extends GenericDAO;
Thread safety:
Implementations of this interface should be thread safe.

ClientStatusDAO<ClientStatus, Long> <<interface>>
Usage:
This interface represents the ClientStatusDAO business interface.
This interface defines the specific methods available for the ClientStatusDAO business interface: get the clients with the corresponding status.
See base class for other available operations.
Implementation notes:
extends GenericDAO;
Thread safety:
Implementations of this interface should be thread safe.

ProjectStatusDAO<ProjectStatus, Long> <<interface>>
Usage:
This interface represents the ProjectStatusDAO business interface.
This interface defines the specific methods available for the ProjectStatusDAO business interface: get the projects with the corresponding status.
See base class for other available operations.
Implementation notes:
extends GenericDAO;
Thread safety:
Implementations of this interface should be thread safe.

ClientDAO<Client, Long> <<interface>>
Usage:
This interface represents the ClientDAO business interface.
This interface defines the specific methods available for the ClientDAO business interface: get the projects for the corresponding client.
See base class for other available operations.
Implementation notes:

extends GenericDAO;
Thread safety:
Implementations of this interface should be thread safe.

1.4.3 com.topcoder.clients.dao.ejb3

ClientDAOLocal<<interface, @Local>>
Usage:
This interface represents the ClientDAOLocal local interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the local interface.
Annotations:
@Local
Implementation notes:
extends ClientDAO;
Thread safety:
Implementations of this interface should be thread safe.

ProjectStatusDAOLocal<<interface, @Local>>
Usage:
This interface represents the ProjectStatusDAOLocal local interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the local interface.
Annotations:
@Local
Implementation notes:
extends ProjectStatusDAO;
Thread safety:
Implementations of this interface should be thread safe.

ClientStatusDAOLocal<<interface, @Local>>
Usage:
This interface represents the ClientStatusDAOLocal local interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the local interface.
Annotations:
@Local
Implementation notes:
extends ClientStatusDAO;
Thread safety:
Implementations of this interface should be thread safe.

CompanyDAOLocal<<interface, @Local>>
Usage:
This interface represents the CompanyDAOLocal local interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the local interface.
Annotations:
@Local
Implementation notes:
extends CompanyDAO;
Thread safety:
Implementations of this interface should be thread safe.

ProjectDAOLocal<<interface, @Local>>
Usage:

This interface represents the ProjectDAOLocal local interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the local interface.
Annotations:
@Local
Implementation notes:
extends ProjectDAO;
Thread safety:
Implementations of this interface should be thread safe.

ClientDAORemote<<interface, @Remote>>
Usage:
This interface represents the ClientDAORemote remote interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the remote interface.
Annotations:
@Remote
Implementation notes:
extends ClientDAO;
Thread safety:
Implementations of this interface should be thread safe.

ProjectStatusDAORemote<<interface, @Remote>>
Usage:
This interface represents the ProjectStatusDAORemote remote interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the remote interface.
Annotations:
@Remote
Implementation notes:
extends ProjectStatusDAO;
Thread safety:
Implementations of this interface should be thread safe.

ClientStatusDAORemote<<interface, @Remote>>
Usage:
This interface represents the ClientStatusDAORemote remote interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the remote interface.
Annotations:
@Remote
Implementation notes:
extends ClientStatusDAO;
Thread safety:
Implementations of this interface should be thread safe.

CompanyDAORemote<<interface, @Remote>>
Usage:
This interface represents the CompanyDAORemote remote interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the remote interface.
Annotations:
@Remote

Implementation notes:
extends CompanyDAO;
Thread safety:
Implementations of this interface should be thread safe.

ProjectDAORemote<<interface, @Remote>>
Usage:
This interface represents the ProjectDAORemote remote interface of the session bean.
See base class for available operations.
Defines a static String variable containing the JNDI name of the remote interface.
Annotations:
@Remote
Implementation notes:
extends ProjectDAO;
Thread safety:
Implementations of this interface should be thread safe.

ProjectDAOBean<<class, @Stateless>>
Usage:
This class is a Stateless Session Bean realization of the ProjectDAO business interface.
This class has a default no-arg constructor.
This class implements the method available for the ProjectDAO business interface: retrieve project by id and retrieve all projects.
See base class for other available operations.
It uses the EntityManager configured in the base class to perform the needed operations, retrieve the EntityManager using its corresponding getter.
Annotations:
@Stateless(name=ProjectDAO.BEAN_NAME)
@Local(ProjectDAOLocal.class)
@Remote(ProjectDAORemote.class)
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
Implementation notes:
extends GenericDAOEJB3DAO
implements ProjectDAO,ProjectDAOLocal,ProjectDAORemote

Changes in Configurable Contest Fees v1.0 Assembly, the following methods are added:
- searchProjectsByClientName : Search projects by client name
- searchProjectsByProjectName : Search projects by project name.
- saveContestFees : Save contest fees.
- getContestFeesByProject : get contest fees by project.

Changes in v1.1:
The following methods are implemented according to interface changes of ProjectDAO:
- addUserToBillingProjects: Adds the user with given user name to the billing projects that are specified with project IDs.
- removeUserFromBillingProjects: Removes the user with given user name from the billing projects that are specified with project IDs.
- getProjectsByClientId: Get all projects associated with the given client id.

In version 1.2, this classes adds two new method, updateProjectBudget updates the project's budget, getUsersByProject gets all users' names who have access of the billing project.

Thread safety:
This class is technically mutable since the inherited configuration properties (with

@PersistenceContext) are set after construction, but the container will not initialize the properties more than once for the session beans and the EJB3 container ensure the thread safety in this case.

CompanyDAOBean<<class, @Stateless>>
Usage:
This class is a Stateless Session Bean realization of the CompanyDAO business interface.
This class has a default no-arg constructor.
This class implements the method available for the CompanyDAO business interface: retrieve clients for company and retrieve projects for company.
See base class for other available operations.
It uses the EntityManager configured in the base class to perform the needed operations, retrieve the EntityManager using its corresponding getter.
Annotations:
@Stateless(name=CompanyDAO.BEAN_NAME)
@Local(CompanyDAOLocal.class)
@Remote(CompanyDAORemote.class)
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
Implementation notes:
extends GenericDAOEJB3DAO
implements CompanyDAO,CompanyDAOLocal,CompanyDAORemote;
Thread safety:
This class is technically mutable since the inherited configuration properties (with @PersistenceContext) are set after construction, but the container will not initialize the properties more than once for the session beans and the EJB3 container ensure the thread safety in this case.

ClientStatusDAOBean<<class, @Stateless>>
Usage:
This class is a Stateless Session Bean realization of the ClientStatusDAO business interface.
This class has a default no-arg constructor.
This class implements the method available for the ClientStatusDAO business interface: get the clients with the corresponding status.
See base class for other available operations.
It uses the EntityManager configured in the base class to perform the needed operations, retrieve the EntityManager using its corresponding getter.
Annotations:
@Stateless(name=ClientStatusDAO.BEAN_NAME)
@Local(ClientStatusDAOLocal.class)
@Remote(ClientStatusDAORemote.class)
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
Implementation notes:
extends GenericDAOEJB3DAO
implements ClientStatusDAO,ClientStatusDAOLocal,ClientStatusDAORemote;
Thread safety:
This class is technically mutable since the inherited configuration properties (with @PersistenceContext) are set after construction, but the container will not initialize the properties more than once for the session beans and the EJB3 container ensure the thread safety in this case.

ProjectStatusDAOBean<<class, @Stateless>>
Usage:

This class is a Stateless Session Bean realization of the ProjectStatusDAO business interface.
This class has a default no-arg constructor.
This class implements the method available for the ProjectStatusDAO business interface: get the projects with the corresponding status.
See base class for other available operations.
It uses the EntityManager configured in the base class to perform the needed operations, retrieve the EntityManager using its corresponding getter.
Annotations:
@Stateless(name=ProjectStatusDAO.BEAN_NAME)
@Local(ProjectStatusDAOLocal.class)
@Remote(ProjectStatusDAORemote.class)
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
Implementation notes:
extends GenericDAOEJB3DAO
implements ProjectStatusDAO,ProjectStatusDAOLocal,ProjectStatusDAORemote;
Thread safety:
This class is technically mutable since the inherited configuration properties (with @PersistenceContext) are set after construction, but the container will not initialize the properties more than once for the session beans and the EJB3 container ensure the thread safety in this case.

ClientDAOBean<<class, @Stateless>>
Usage:
This class is a Stateless Session Bean realization of the ClientDAO business interface.
This class has a default no-arg constructor.
This class implements the method available for the ClientDAO business interface: get the projects for the corresponding client.
See base class for other available operations.
It uses the EntityManager configured in the base class to perform the needed operations, retrieve the EntityManager using its corresponding getter.
Annotations:
@Stateless(name=ClientDAO.BEAN_NAME)
@Local(ClientDAOLocal.class)
@Remote(ClientDAORemote.class)
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
Implementation notes:
extends GenericDAOEJB3DAO
implements ClientDAO,ClientDAOLocal,ClientDAORemote;
Thread safety:
This class is technically mutable since the inherited configuration properties (with @PersistenceContext) are set after construction, but the container will not initialize the properties more than once for the session beans and the EJB3 container ensure the thread safety in this case.

GenericEJB3DAO<T extends AuditableEntity, Id extends Serializable>
<<abstract, class>>
Usage:
This abstract class represents the base GenericEJB3DAO class.
This base class implements the generic methods available for all the concrete DAOs: retrieve entity by id, retrieve all entities, search entities by name, search entities using the given filter, save a given entity, and delete a given entity.
It is configured with an EntityManager (initialized by the EJB container through dependency injection) needed to perform operations on the persistence and also uses

TopCoder Search Builder 1.4 component to perform the search using the given Filter.
NOTE: it is not an Stateless session bean.
Implementation notes:
implements GenericDAO;
Thread safety:
Implementations of this interface should be thread safe.

SearchByFilterUtility<T extends AuditableEntity, Id extends Serializable> <<interface>>
Usage:
This interface represents the SearchByFilterUtility interface.
This interface defines the method needed to search entities using a given filter as criteria.
Thread safety:
Implementations of this interface should be thread safe.

SearchByFilterUtilityImpl<T extends AuditableEntity, Id extends Serializable> <<class>>
Usage:
This class is a realization of the SearchByFilterUtility interface.
This class implements the method available for the SearchByFilterUtility interface: search entities using the given filter.
Search Builder TopCoder component is used to perform the search operation.
Implementation notes:
implements SearchByFilterUtility;
Thread safety:
This class is stateless, its field is immutable. Search Builder used is used only local and not shared between multiple threads so this component is thread safe.

1.5 Component Exception Definitions

1.5.1 com.topcoder.clients.dao

DAOException<<exception, @ApplicationException>>
USAGE:
This exception is the base exception for all exceptions raised from operations performed on entities (in this design, entities inherited from AuditableEntity) from this design (excerpt loading of the configurations).
This exception wraps exceptions raised from persistence, from usage of the J2EE utilities or used TopCoder components.
ANNOTATIONS:
@ApplicationException(rollback=true) - to notify the EJB container to roll back the transaction.
IMPLEMENTATION NOTES:
extends BaseCriticalException;
THREAD SAFETY:
This exception is not thread safe because parent exception is not thread safe.
The application should handle this exception in a thread-safe manner.

EntityNotFoundException<<exception, @ApplicationException>>
USAGE:
This exception signals an issue if the needed entity (in this design, entities inherited from AuditableEntity) can not be found.
Has an entity id argument in each constructor and a getter for this property.
ANNOTATIONS:
@ApplicationException(rollback=true) - to notify the EJB container to roll back the transaction.
IMPLEMENTATION NOTES:

extends DAOException;
THREAD SAFETY:
This exception is not thread safe because parent exception is not thread safe.
The application should handle this exception in a thread-safe manner.

DAOConfigurationException<<exception, @ApplicationException>>
USAGE:
This runtime exception signals an issue if the configured value is invalid (in this design, when EntityManager is not configured or if it is invalid - null).
Wraps the underlying exceptions when using the configured values.
ANNOTATIONS:
@ApplicationException(rollback=true) - to notify the EJB container to roll back the transaction.
IMPLEMENTATION NOTES:
extends BaseRuntimeException;
THREAD SAFETY:
This exception is not thread safe because parent exception is not thread safe.
The application should handle this exception in a thread-safe manner.

1.5.2 java.lang.

IllegalArgumentException <<exception>>
This system exception is thrown if an argument is invalid (null or long ids <= 0), empty string etc.

1.6 Thread Safety

This component is thread-safe due to the following reasons:

The EJB DAO implementations are technically mutable since the EntityManager configuration property (with @PersistenceContext or with setter) is set after construction, but the container will not initialize the property more than once for the session beans and the container ensure the thread safety in this case.

All modifying methods are transactional managed by the container so EJB DAO implementations can be safely accessed from multiple threads in EJB container.

In version 1.2, the thread safety discussion above still applies.

2. Environment Requirements

2.1 Environment

- Sun Java 5.0 SE runtime version 1.5.0 is required for development and compilation
- RedHat Linux 7
- Solaris 7
- Informix 10
- EJB3
- J2EE

2.2 TopCoder Software Components

- **Base Exception 2.0** – it is used as base exception for custom exceptions created in this design.
- **Search Builder 1.4** - is used to perform HQL search of the entities corresponding to the given Filter.
- **Configuration API 1.0:** This is used as the main abstraction for configuration.
- **Configuration Persistence 1.0.1:** This will be used as the actual specific configuration reader for this component.

- **Object Factory 2.1:** This is used to instantiate configured objects as needed by this component.
- **Object Factory Configuration Manager Plugin 1.0:** This is used by object factory 2.1.
- **Object Factory Configuration API Plugin 1.0:** This is used by object factory 2.1.

ID Generator 3.0: Used to generate user account id.

2.3 Third Party Components

- None

3. Installation and Configuration

3.1 Package Name

- **com.topcoder.clients.model**– this package contains all the entities required for this design and their base class.
- **com.topcoder.clients.dao** – contains all the DAO interfaces required for this design. Also the needed custom exceptions are placed here.
- **com.topcoder.clients.dao.ejb3** – contains all the DAO stateless a session beans implementations and their base class. Also for each non generic DAO interface, a remote and a local interface are defined and search by filter utility.

3.2 Configuration Parameters

3.2.1 Persistence Context

Configuration of EntityManager needed in beans is realized using @PersistenceContext (unitName="persistenceUnit", type=PersistenceContextType.TRANSACTION):

Parameter	Description	Values
unitName	Represents persistence context name to obtain the EntityManager instance from the container. ***Required***.	**String**. Can not be null or empty.

3.2.2 Resources

Configuration of resources needed in beans is realized using @Resource(name = resourceName):

Parameter	Description	Values
search_bundle_manager_namespace	Represents the search bundle manager namespace. ***Required***.	**String**. Can not be null or empty.
config_file_name	Represents the configuration file name. ***Required***.	**String**. Can not be null or empty.
config_namespace	Represents the configuration namespace. ***Required***.	**String**. Can not be null or empty.

3.2.3 Configuration properties inside the ConfigurationObject

Parameter	Description	Values
search_by_filter_utility_token	Represents the token used to get the SearchByFilterUtility instance via object	**String**. Can not be null or

	factory. ***Required***.	empty.

3.3 Dependencies Configuration

- EJB 3.0 and Search Builder 1.4 should be properly configured.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Prepare the database by executing test_files/Schema.sql
- Modify the connection strings in test_files/hibernate.cfg.xml and test_files/META-INF/informix-ds.xml
- Install and start JBoss AS.
- Run ant test.setup to deploy the component into JBoss container.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

- The entities, interfaces and beans should be deployed in an EJB 3.0 container.

4.3 Demo

The following demo shows the usage of the operations available over the Project. Operations available over the other entities from this design are quite similar.

4.3.1 Demo for version 1.0

```
// retrieve bean
InitialContext ctx = new InitialContext();
ProjectDAORemote bean = (ProjectDAORemote) ctx
        .lookup("client_project_entities_dao/ProjectDAOBean/remote");

Filter filter = new EqualToFilter("projectStatus", project
        .getProjectStatus().getId());

getEntityManager().close();

List<Project> projects;

// get project for corresponding id
Project tempProject = bean.retrieveById(100L);

// get all projects
projects = bean.retrieveAll();

// get all projects with the name "name"
projects = bean.searchByName("name");

// get all that match the given filter
projects = bean.search(filter);

// save or update a project
bean.save(project);

// delete the project
bean.delete(project);
```

```
// get project for corresponding id without projectChildren
tempProject = bean.retrieveById(100L, false);

// get project for corresponding id with projectChildren
tempProject = bean.retrieveById(100L, true);

// get all projects without projectChildrens
projects = bean.retrieveAll(false);
```

4.3.2 Demo for updates between version 1.0 and 1.1

```
// We use the bean created in 4.3.1
// get projects by user
projects = bean.getProjectsByUser("username");

// get all projects only
projects = bean.retrieveAllProjectsOnly();

// search projects by project name
projects =
        bean.searchProjectsByProjectName("projectname");

// search projects by client name
projects = bean.searchProjectsByClientName("clientname");

// get contest fees by project
List<ProjectContestFee> fees =
          bean.getContestFeesByProject(100L);

// save contest fees
bean.saveContestFees(fees, 100L);

// check client project permission
boolean clientProjectPermission =
        bean.checkClientProjectPermission("username", 100L);

// check po number permission
boolean poNumberPermission =
        bean.checkPoNumberPermission("username", "123456A");
```

4.3.3 Demo for version 1.1

```
// We use the bean created in 4.3.1
// Add user to billing projects
bean.addUserToBillingProjects("username",
        new long[] {100L, 101L, 102L});

// Remove user from billing projects
bean.removeUserFromBillingProjects("username",
        new long[] {100L, 101L});


// get projects by client id
projects = bean.getProjectsByClientId(200L);
```

4.3.4 Demo for version 1.2

Suppose project table has the following record:

project_id	budget
100	4000

```
// Increase the budget of project with id 100 with 500.
double newBudget = bean.updateProjectBudget(”ivern”,100,500);

/* 4500 should be returned and the budget for id 100 will be updated to 4500,
besides, a record will be added into project_budget_audit table with project_id =
100, changed_amount = 500 and creation_user=”ivern”. */
```

Suppose user_account table has the following records:

user_account_id	user_name
101	user1
102	user2
103	user3
104	user4
105	user5

and project_manager table has the following records:

active	project_id	user_account_id
1	123	101
0	123	102

and project_worker table has the following records(suppose for all records the current date is between start date and end date):

active	project_id	user_account_id
1	123	104
0	123	105

```
// Now, get the user names who have access of the billing project with id 123.
List<String> users = bean.getUsersByProject(123)
// users list should contain 2 elements(“user1” and “user4”).
```

5. Future Enhancements

- Further DAO implementations will be added.