Client Associations Services 1.0 Component Specification

1. Design

This component will provide services for the following associations:

- Clients and components: meaning that a component (or application) is developed for the client.
- Clients and users: meaning that a user is authorized to work with the client components.

The services provided by this component will enable the creation, deletion and query of those associations, using database persistence.

Design considerations and overview:
As required the operations will be provided through a stateless session bean with both the remote and local interfaces defined, and the persistence must be implemented using Hibernate.

There are two possible solutions to meet the requirements:

- Use Hibernate via JPA interface in the stateless session bean.
- Use Hibernate to implement a DAO interface whose implementations are pluggable to the stateless session bean.

Both solutions are acceptable as confirmed by the architect. This design uses the second solution because it properly separates the responsibilities of a stateless session bean and a persistence implementation which makes the whole component very flexible and easy to test; besides it also makes the DAO interfaces and implementations highly reusable in other frameworks such as Spring or even a normal application that doesn't depend on any application framework but only uses Hibernate as persistence tool.

The Hibernate configuration file, the Hibernate mapping files and the auto generated classes are provided in the docs\src directory as required.

1.1 Design Patterns

Strategy Pattern is used to make ClientAssociationDAO implementations pluggable to the ClientAssociationServiceStatelessSessionBean class.

DAO Pattern is used to provide creation, deletion and query of the required associations.

We could say that the **Delegate Pattern** is used since ClientAssociationServiceStatelessSessionBean delegates all tasks to an underlying ClientAssociationDAO instance.

1.2 Industry Standards

EJB 3
ORM

1.3 Required Algorithms

There is no complicated algorithm used in this component; all necessary details are present in TC UML Tool documentation tabs.

1.4 Component Class Overview

ClientAssociationServiceRemote [interface]:
This class defines all the required methods to manage various associations. Implementations of it should be thread safe.

ClientAssociationServiceRemote [interface]:

This is the EJB remote interface for the stateless session bean. It defines all the required methods to manage various associations. It must have the "@Remote" annotation. This interface is supposed to be used by classes/applications that don't stay in the same JVM as the EJB container, generally this means invocation via RMI.
Implementations of it should be thread safe.

ClientAssociationServiceLocal [interface]:
This is the EJB local interface for the stateless session bean. It defines all the required methods to manage various associations. It must have the "@Local" annotation. This interface is supposed to be used by classes/applications that stay in the same JVM as the EJB container, such as a servlet.
Implementations of it should be thread safe.

ClientAssociationServiceStatelessSessionBean [class]:
This class is an implementation of the remote and local interfaces and is implemented as a stateless session bean. It uses a ClientAssociationDAO instance to perform all necessary persistence, basically this class simply delegates all tasks to the DAO instance.
This class must have the following annotations:

    @Stateless
    @TransactionManagement(TransactionManagementType.CONTAINER)

This class is thread safe since it's immutable and all operations are transaction safe.

ClientAssociationDAO [interface]:
This interface defines the persistence contract for the association operations. It enables the creation, deletion and query of those associations. Implementations of this interface may or may not handle transactions directly depending on whether the session bean using it uses CMT or BMT.
Implementations of this interface may not be thread safe if intended to be used with session bean using CMT (since they don't handle transactions themselves), but they are used in a thread safe way. If the implementations are not intended to be used with session bean using CMT, then the implementations must be thread safe.

ClientAssociationHibernateDAO [class]:
This class implements the ClientAssociationDAO interface and uses a database as storage media. This implementation completely bases on the Hibernate framework and can work with most popular databases by providing proper configuration. This class uses some classes generated from the given database schema to perform all necessary database operations. This class itself doesn't use transaction directly, but the EJB container will handle all the transaction logic.
Strictly speaking this class is not thread safe since although it is immutable, the mutating operations are not self transaction safe, but it’s expected the session bean using this DAO will use CMT, which means all transaction management will be done by the EJB container.

HibernateHelper [class]:
This helper class is used to provide a central SessionFactory for the whole application (like a singleton SessionFactory) as creating a SessionFactory is very expensive and it's safe to share a SessionFactory between different threads concurrently.
This class is thread safe as it's immutable.

Client [class]:
This class is auto generated from the Hibernate tool. It corresponds to the Client table in the DDL. Please consult the generated source code for more details.
This class is mutable so not thread safe.

CompClient [class]:

This class is auto generated from the Hibernate tool. It represents the relationship between a component and a client, a combination of it and the CompClientPK class corresponds to the comp_client table in the DDL. Please consult the generated source code for more details.
This class is mutable so not thread safe.

UserClient [class]:
This class is auto generated from the Hibernate tool. It represents the relationship between a user and a client, a combination of it and the UserClientPK class corresponds to the user_client table in the DDL. Please consult the generated source code for more details.
This class is mutable so not thread safe.

CompClientPK [class]:
This class is auto generated from the Hibernate tool. It represents the primary key of the comp_client table, a combination of it and the CompClient class describes the whole relationship between a component and a client and corresponds to the comp_client table in the DDL. Please consult the generated source code for more details.
This class is mutable so not thread safe.

UserClientPK [class]:
This class is auto generated from the Hibernate tool. It represents the primary key of the user_client table, a combination of it and the UserClient class describes the whole relationship between a user and a client and corresponds to the user_client table in the DDL. Please consult the generated source code for more details.
This class is mutable so not thread safe.

1.5 Component Exception Definitions

ClientAssociationServiceException:
This exception is created to represent any errors that might occur in the methods defined in the session beans, for example a persistence error. This exception is thrown by the methods in the session beans.

This exception must have the "@ApplicationException(rollback=true)" annotation so that if any error occurs the transaction could be automatically rolled back by EJB container.

ClientAssociationDAOException:
This exception is created to represent any errors that might occur in the persistence layer, for example: a connection error, database down, etc... This exception is thrown by the methods in ClientAssociationDAO interface and its implementations.

This exception must have the "@ApplicationException(rollback=true)" annotation so that if any error occurs the transaction could be automatically rolled back by EJB container.

1.6 Thread Safety

Strictly speaking this component is not thread safe as it contains some mutable classes and ClientAssociationDAO implementations might not be transaction safe. However, this component could be used in a thread safe way.

The EJB interfaces require their implementations to be thread safe. And the DAO interface either requires its implementations to be thread safe (if intended to be used with BMT) or expects the EJB container to manage the transaction (when used with CMT). The stateless session bean is immutable, all operations in it are transaction safe so it’s thread safe. The DAO interface implementation, though not transaction safe, is immutable and used by a session bean using CMT, so there won’t be threading issues

when being used.

Instances of the mutable classes are only created and used internally and thus will never be mutable by external classes when being used, as such they are always used in a thread safe way in this component.

So in a word, this component could be considered completely thread safe when being used, since the EJB container will properly manage transactions and no external threads will be able to unexpectedly modify the state of this component. There is no need (and also impossible) to make the component completely thread safe because:
- Synchronizing all access to the mutable classes is a big overkill.
- Making the DAO implementation self transaction safe will cause exceptions since transaction is already handled by the EJB container.

2. Environment Requirements

2.1 Environment

- Development language: Java 5.0
- Compile target: Java 5.0
- RedHat Enterprise Linux 4
- JBoss 4.2 GA with EJB 3.0
- Informix 10.00.UC 5

2.2 TopCoder Software Components

- **JNDI Context Utility 2.0** is used to look up DAO implementations from JNDI tree.
- **Base Exception 2.0** is used as the base of the custom exceptions in this component.
- **Configuration Manager 2.1.5** is used by **JNDI Context Utility 2.0**.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

- **Hibernate 3.2.5** is used to perform the actual persistence jobs. You can get Hibernate from http://www.hibernate.org/. You may also find the Hibernate Tools Plugin and the Middlegen Tool useful as they could be used to generate java code from hbm file and generate hbm file from DDL respectively. Both tools could be get on the hibernate site.

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.topcoder.controlpanel.clientassociations
com.topcoder.controlpanel.clientassociations.dao
com.topcoder.controlpanel.clientassociations.dao.hibernate

3.2 Configuration Parameters

This component doesn't employ any custom configuration strategies. The only thing to mention is if you don't want to use the default ClientAssociationHibernateDAO implementation, you need to bind a custom implement under the JNDI name "dao/clientAssociationDAO" in the JNDI tree.

3.3 Dependencies Configuration

If applicable, all dependent components should be properly configured before this component is used. Please consult the documents of the dependent components for configuration details.

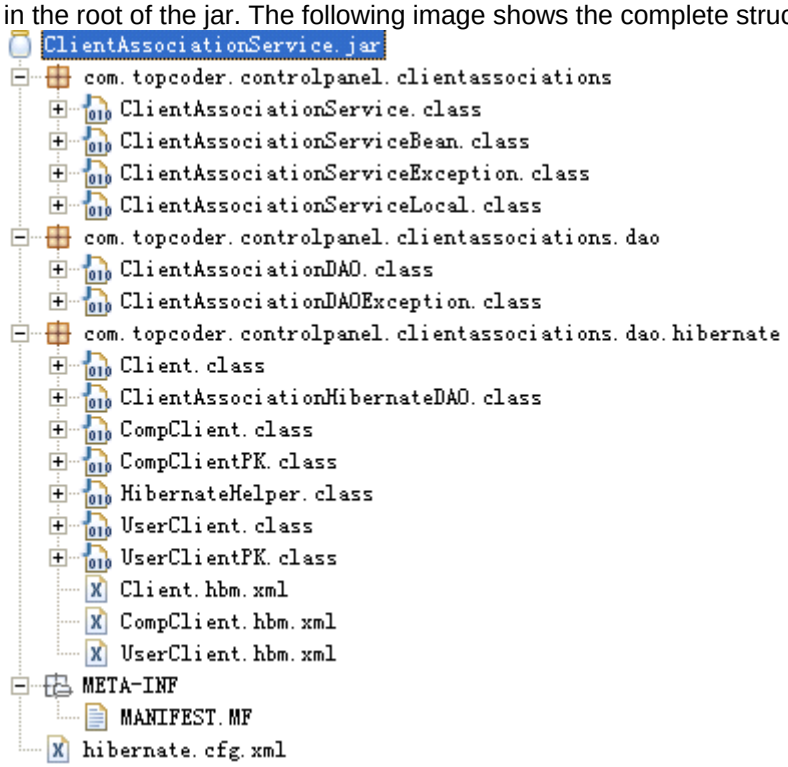
4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

- Package this component into a jar, note the *.hbm.xml files must be in the same package as the classes generated from them; besides, hibernate.cfg.xml must be put in the root of the jar. The following image shows the complete structure of the jar:



- Copy the TopCoder libraries and Hibernate libraries (if JBoss doesn't have them) to %JBOSS_HOME%/server/default/lib directory.
- Copy ClientAssociationService.jar to %JBOSS_HOME%/server/default/deploy directory.
- Start JBoss and then follow demo.

4.3 Demo

First we need to get an instance of the session bean:

```
// create a new context
InitialContext ctx = new InitialContext();
// look up the session bean
ClientAssociationServiceRemote cas = (ClientAssociationServiceRemote) ctx
               .lookup("ClientAssociationServiceStatelessSessionBean");
```

Another simpler way to do this is define a field in the following way:

```
@EJB
ClientAssociationServiceRemote cas;
```

When the @EJB annotation is seen when the class is loaded, the container (for example: a servlet container, in which case the above code is used in a servlet) looks up the ClientAssociationServiceRemote EJB behind the scenes and sets the cas variable to the retrieved EJB reference. If necessary, the container will lookup the EJB remotely over RMI. The @EJB annotation works in any component that is registered with the Java EE container such as a Servlet or JSF backing bean. As long as you are using the standard Java EE stack, this is probably more than sufficient.

Now that we now have an instance of the session bean, we can start using the methods. But before we do that lets assume the database has the following records:
client table:

client_id	name
1	Client1
2	Client2
3	Client3

comp_client table:

component_id	client_id
1	1
2	1
3	3

user_client table:

user_id	client_id	admin_ind
1	2	1
2	3	0

```
// assign a component
cas.assignComponent(4, 1);
// as we can expect, the comp_client table now has 4 records
```

component_id	client_id
1	1
2	1
3	3
4	1

```
// calling the above method twice would cause exception though since the record
// already exists, this also applies to the assignUser method

// assign a user
cas.assignUser(2l,1,true);
// the user_client table now has 3 records
```

user_id	client_id	admin_ind
1	2	1
2	3	0
2	1	1

```
// however if you do this, you'll get an exception since the client doesn't exist,
// this also applies to the other mutating methods
cas.assignUser(2,4,false);
```

```
// get components by client
// this returns a list with 3 elements: [1,2,4]
List<Long> compIds = cas.getComponentsByClient(1);

// or we could get components by user (get components from all clients to which
// the user is assigned)
// this returns a list with 4 elements: [1,2,3,4]
compIds = cas.getComponentsByUser(2);

// check if a user is an admin user
// this will return true
boolean isAdmin = cas.isAdminUser(1, 2);
// however this causes exception since there is no association between user 3 and
// client 1
isAdmin = cas.isAdminUser(3, 1);

// remove a component
cas.removeComponent(1, 1);
// now the comp_client table has 3 records again:
```

component_id	client_id
2	1
3	3
4	1

```
// usage of the other operations are just similar so no duplication is provided
// here
```

5. Future Enhancements

None.