[TopCoder]

Client Project Management Services 1.0 Component Specification

1. Design

This component provides a wrapper to the Client and Project Management component, exposing the main management functionality as web services. This component also provides the client classes that allows for easy remote access to the service methods. This component addresses the Client, Project, and Company management services, as well as the authorization pieces associated with those services.

The web service operations perform logging using Logging Wrapper component and authentications and authorizations are performed using JBoss Login Module component.

In addition, a super user (administrator) is added that can perform all the operations without any interactions with UserMappingRetriever.

1.1 Design Patterns

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

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

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

Proxy Pattern – is used by the clients of the implemented web services. The returned interfaces of the services provide the service operations on the remote machine. `XXXServiceClient` client classes actually acts as a Proxy Pattern.

Delegation pattern – the service beans delegates their operations to the configured managers.

1.2 Industry Standards

- EJB 3.0
- JPA 1.0
- JAX-WS 2.0

1.3 Required Algorithms

There are no complicated algorithms in this design.

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 entities from this design extend from AuditableEntity Client Project Entities DAO component.

1.3.2 Logging

1.3.2.1 *The Log field can be initialized using:*

If a logName is provided (not null or empty String)

```
log = LogManager.getLog( logName )
```

or:

`log = LogManager.getLog()` if no logName is provided

Logging is used in this way:

```
logger.log( Level.DEBUG, "Some internal details" );
```

1.3.2.2 *Logging can be turned on or off using:*

`setLog(newLog)` method. If `newLog` is null, logging is disabled. To enable logging, a valid value should be provided.

1.3.2.3 *Determine the type of the logging:*

```
//Determine if logging should be performed:
if( !log ==null)
       // determine the type of logging:
       if (verboseLogging== true)
       //detailed logging actions should be performed
       else
       //standard logging should be performed
else.
//if false: no logging should be performed.
```

1.3.2.4 *There are two types of logging that can be set using:*

setVerboseLogging(verboseLogging:boolean) method:

- ***Standard log* that logs only the following:**

  `if(getVerboseLogging()==false):`
  - Method entry and parameter information (`Level.DEBUG` level);
  - Any exception `(Level.WARNING level)` before the exception is re thrown. The exception name and stack trace should be logged.
- ***Detailed log*** that logs exceptions and other details:

  `if(getVerboseLogging()==true):`
  - Entering the method and timestamp (`Level.DEBUG level`);
  - Method arguments (`Level.DEBUG level`);
  - Time spent in the method (`Level.DEBUG level`);
  - Return value (`Level.DEBUG level`). Any exception `(Level.WARNING level)` before the exception is re thrown. The exception name and stack trace should be logged.
  - Any exception `(Level.WARNING level)` before the exception is re thrown. The exception name and stack trace should be logged.

1.3.3 Initialize the needed dependencies

This section shows the initialization of the needed dependencies for the stateless session's beans. The dependencies should be initialized using the configured resources and expected tokens:

dependency	fileResource	namespaceResource	token
projectManager	projectManagerFile	projectManagerNamespace	project_manager_token
clientManager	clientManagerFile	clientManagerNamespace	client_manager_token
companyManager	company ManagerFile	company ManagerNamespace	company_manager_token
userMapping Retriever	userMappingRetriever File	userMappingRetriever Namespace	user_mapping_ retriever_token

Depending of the needed **dependency**, replace the **fileResource**, **namespaceResource** and **token** with the values from the table from above.

```
// create an ConfigurationFileManager from the given file:
ConfigurationFileManager configurationFileManager = new
ConfigurationFileManager(fileResource);

// create an ConfigurationObject using the give namespace:
ConfigurationObject configurationObject =
configurationFileManager.getConfiguration(namespaceResource);

// get the child ConfigurationObject using the created configurationObject:
configurationObject = configurationObject.getChild(namespaceResource);

// create an ConfigurationObjectSpecificationFactory from the created
// configurationObject:
ConfigurationObjectSpecificationFactory configurationObjectSpecificationFactory =
new
ConfigurationObjectSpecificationFactory(configurationObject);

// create an ObjectFactory from the created
// configurationObjectSpecificationFactory:
ObjectFactory objectFactory = new
ObjectFactory(configurationObjectSpecificationFactory);

// get the needed token value from the created configurationObject:
String tokenValue = configurationObject.getPropertyValue(token);

// create the needed dependency using the created objectFactory and the
// retrieved token value:
dependency = (DependencyClass)objectFactory.createObject(tokenValue);
```

1.3.4 *Authorization and authentication*

The services defined in this component relies on JAAS authentication of users to be performed under control of the EJB container, and it relies on the security roles configured for the user in making authorization decisions.
Two security roles will be used: "User" and "Admin".
These need to be declared either in the bean's deployment descriptor or via annotation of the bean class. The annotation to use is:

```
@DeclareRoles({ Roles.USER, Roles.ADMIN })
@RolesAllowed({ Roles.USER, Roles.ADMIN })
```

- *Case 1: user is administrator:*

Means that the user has full privileges and no other authentication should be used and the user is authorized to perform the given operation:

```
// check whether the caller is an administrator
if(sessionContext.isCallerInRole(administratorRole))
// do the rest of the operation
else check if the user is client and project user (case 2)
```

- *Case 2: user is client and project user:*

In some cases, the bean implementation must obtain the user's ID in order to make fine-grained authorization decisions. For this purpose the bean must obtain the Principal object identifying the caller, which it assumes to be of type UserProfilePrincipal (on account of the TopCoder JBoss Login Module being in use). The principal is available from the bean's session context. The complete procedure is as follows:

```
// check whether the caller is an clientAndProjectUserRole role
```

```
if(sessionContext.isCallerInRole(clientAndProjectUserRole))
// do the rest of the operation
else throw an exception (case 3)

// obtain the user profile principal:
UserProfilePrincipal principal = (UserProfilePrincipal)
sessionContext.getCallerPrincipal();

// obtain the user ID
long userId = principal.getUserId();
```

Now if the user operates on some **project**, check if the user is associated to the given **project**:

```
List<Long> validUsersIdsForProject =
userMappingRetriever.getValidUsersForProject(project);

if (validUsersIdsForProject.contains(userId))
// do the rest of the operation
else throw an exception (case 3)
```

Now if the user operates on some **client**, check if the user is associated to the given **client**:

```
List<Long> validUsersIdsForClient =
userMappingRetriever.getValidUsersForClient(client);

if (validUsersIdsForClient.contains(userId))
// do the rest of the operation
else throw an exception (case 3)
```

- *Case 3: user authentication failed:*

If authentication failed means the user is not authorized to perform this operation and:

```
throw new AuthorizationFailedException(..).
```

1.3.5 Update the user and date properties for the entities

In this design, entity means a subclass of AuditableEntity: Client, Project and Company.

- *Create entity:*

Update Dates: when creating an entity, the entity.createDate and entity.modifyDate should be set to the current date using appropriate setCreateDate(new Date()) and setModifyDate(new Date())methods.

Update Users: when creating an entity, the entity.createUsername and entity.modifyUsername should be set to the current user using appropriate setCreateUsername(**principal**.getName()) and setModifyUsername(**principal**.getName())methods.

- *Other operations over the entity like delete, update or set some properties:*

Update Dates: the entity.modifyDate should be set to the current date using appropriate setModifyDate(new Date()) method.

Update Users: the entity.modifyUsername should be set to the current user using appropriate setModifyUsername(**principal**.getName())method.

Note: please note the fact that the entities are not really deleted from the database, only marked as deleted using the is_deleted flag. So the modifyDate and modifyUsername should be updated too.

1.4 Component Class Overview

1.4.1 com.topcoder.clients.webservices

ProjectService
This interface represents the ProjectService web service endpoint interface.
This interface defines the method available for the ProjectService web service: create, update and delete project, set project status for project.

ClientService
This interface represents the ClientService web service endpoint interface.
This interface defines the method available for the ClientService web service: create, update and delete client, set code name for client and set client status for client.

CompanyService
This interface represents the CompanyService web service endpoint interface.
This interface defines the methods available for the CompanyService web service: create, update and delete company.

ProjectServiceLocal
This interface represents the ProjectService local interface of the session bean.
Also it defines a static String variable containing the JNDI name of the local interface.
See base interface for all the available operations.

ClientServiceLocal
This interface represents the ClientService local interface of the session bean.
Also it defines a static String variable containing the JNDI name of the local interface.
See base interface for all the available operations.

CompanyServiceLocal
This interface represents the CompanyService local interface of the session bean.
Also it defines a static String variable containing the JNDI name of the local interface.
See base interface for all the available operations.

ProjectServiceRemote
This interface represents the ProjectService remote interface of the session bean.
Also it defines a static String variable containing the JNDI name of the remote interface.
See base interface for all the available operations.

ClientServiceRemote
This interface represents the ClientService remote interface of the session bean.
Also it defines a static String variable containing the JNDI name of the remote interface.
See base interface for all the available operations.

CompanyServiceRemote
This interface represents the CompanyService remote interface of the session bean.
Also it defines a static String variable containing the JNDI name of the remote interface.
See base interface for all the available operations.

1.4.2 com.topcoder.clients.webservices.beans

ProjectServiceBean
This class is a Stateless Session Bean endpoint realization of ProjectService web service interface.
This class has a default no-arg constructor.
It uses ProjectManager from Client Project Management component by delegating to namesake methods to perform the operations.
This class implements the methods available for the ProjectService web service: create, update and delete project, set project status for project.
All the available web service operations perform logging using the Logging Wrapper component.

All the available web service operations performs the authentication and authorization of the users that performs the operations using JBoss Login Module component and then delegates to the Client Project Management component which performs the needed operations.

ClientServiceBean
This class is a Stateless Session Bean endpoint realization of ClientService web service interface.
This class has a default no-arg constructor.
It uses ClientManager from Client Project Management component by delegating to namesake methods to perform the operations.
This class implements the methods available for the ClientService web service: create, update and delete client, set code name for client and set client status for client.
All the available web service operations perform logging using the Logging Wrapper component.
All the available web service operations performs the authentication and authorization of the users that performs the operations using JBoss Login Module component and then delegates to the Client Project Management component which performs the needed operations.

CompanyServiceBean
This class is a Stateless Session Bean endpoint realization of CompanyService web service interface.
This class has a default no-arg constructor.
It uses CompanyManager from Client Project Management component by delegating to namesake methods to perform the operations.
This class implements the methods available for the CompanyService web service: create, update and delete company.
All the available web service operations perform logging using the Logging Wrapper component.
All the available web service operations are under the "User" and "Admin" security roles.

1.4.3 com.topcoder.clients.webservices.webserviceclients

ClientServiceClient
This class is a client of ClientService service.
This class has tree constructors that use the wsdlDocumentLocation and service name to create this class instance.
This class has only one method getClientServicePort() that return the service endpoint interface.

ProjectServiceClient
This class is a client of ProjectService service.
This class has tree constructors that use the wsdlDocumentLocation and service name to create this class instance.
This class has only one method getProjectServicePort() that return the service endpoint interface.

CompanyServiceClient
This class is a client of CompanyService service.
This class has tree constructors that use the wsdlDocumentLocation and service name to create this class instance.
This class has only one method getCompanyServicePort() that return the service endpoint interface.

1.4.4 com.topcoder.clients.webservices.usermapping

UserMappingRetriever
This interface represents the UserMappingRetriever interface.
This interface defines the specific methods available for the UserMappingRetriever interface: get valid users for client, get valid users for project, get clients for user id and get projects for user id.

1.4.5 com.topcoder.clients.webservices.usermapping.impl

JPAUserMappingRetriever
This class is a JPA realization of the UserMappingRetriever interface.
This class has a default no-arg constructor and a constructor that receive a entityManager to initialize the namesake field.

This class implements the methods available for the UserMappingRetriever interface: get valid users for client, get valid users for project, get clients for user id and get projects for user id.
It uses the configured EntityManager to perform the needed operations, retrieve the EntityManager using its corresponding getter.

UserClientMapping
This class represents the UserClientMapping java bean. A UserClientMapping can contain a client identifier and a user identifier.
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.

UserProjectMapping
This class represents the UserProjectMapping java bean. A UserProjectMapping can contain a project identifier and a user identifier.
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).

1.5 Component Exception Definitions

1.5.1 com.topcoder.clients.webservices.webserviceclients

ServiceClientCreationException
This runtime exception signals an issue when the creation of the web service client failed (wraps the MalformedURLException when creating the URL given a String wsdl document location). It is not thrown directly in this design; it is just a base class for the concrete service client exceptions.

ClientServiceClientCreationException
This runtime exception signals an issue when the creation of the client web service client failed (wraps the MalformedURLException when creating the URL given a String wsdl document location).

ProjectServiceClientCreationException
This runtime exception signals an issue when the creation of the project web service client failed (wraps the MalformedURLException when creating the URL given a String wsdl document location).

CompanyServiceClientCreationException
This runtime exception signals an issue when the creation of the company web service client failed (wraps the MalformedURLException when creating the URL given a String wsdl document location).

1.5.2 com.topcoder.clients.webservices

ClientProjectManagementServicesException
This exception is the base exception for all exceptions raised from operations from Web Service (excerpt loading of the configurations).
This exception wraps exceptions raised from persistence or from usage of the J2EE utilities or from the used TopCoder components. It is not thrown directly in this design; it is just a base class for the service exceptions.

ClientServiceException
This exception is the base exception for all exceptions raised from operations from ClientService (excerpt loading of the configurations).
This exception wraps exceptions raised from persistence or from usage of the J2EE utilities or from the used TopCoder components.

ProjectServiceException
This exception is the base exception for all exceptions raised from operations from ProjectService (excerpt loading of the configurations).

This exception wraps exceptions raised from persistence or from usage of the J2EE utilities or from the used TopCoder components.

CompanyServiceException
This exception is the base exception for all exceptions raised from operations from CompanyService (excerpt loading of the configurations).
This exception wraps exceptions raised from persistence or from usage of the J2EE utilities or from the used TopCoder components.

1.5.3 com.topcoder.clients.webservices.beans

ServiceBeanConfigurationException
This runtime exception signals an issue if the configured value is invalid (in this design, when managers are null, user mapping retriever is null or required resources are null or empty String). It is not thrown directly in this design; it is just a base class for the concrete service bean configuration exceptions.
Also it wraps the underlying exceptions when using the configured values.

ClientServiceBeanConfigurationException
This runtime exception signals an issue if the configured value is invalid in ClientServiceBean (in this design, when managers are null, user mapping retriever is null or required resources are null or empty String).
Also it wraps the underlying exceptions when using the configured values.

ProjectServiceBeanConfigurationException
This runtime exception signals an issue if the configured value is invalid in ProjectServiceBean (in this design, when managers are null, user mapping retriever is null or required resources are null or empty String).
Also it wraps the underlying exceptions when using the configured values.

CompanyServiceBeanConfigurationException
This runtime exception signals an issue if the configured value is invalid in CompanyServiceBean (in this design, when managers are null, user mapping retriever is null or required resources are null or empty String).
Also it wraps the underlying exceptions when using the configured values.

AuthorizationFailedException
This exception signals an issue when performing the authentications and signals that the given user is not authorized to perform the given operation.

1.5.4 com.topcoder.clients.webservices.usermapping

UserMappingRetrievalException
This exception signals an issue when performing the operations: get valid users for client and get valid users for project.
This exception wraps exceptions raised from persistence or from usage of the J2EE utilities.

EntityNotFoundException
This exception signals an issue if the needed entity (in this design, entities inherited from AuditableEntity) can not be found.

UserNotFoundException
This exception signals an issue if the needed user id can not be found (user id is not mapped with any needed entity: project or client).

1.5.5 java.lang.

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

IllegalStateException

This system exception is thrown if a needed instance is needed and it has not been initialized yet or it has a null value.

1.5.6 javax.xml.ws

WebServiceException
This web service exception is thrown if appears any problem during retrieving the endpoint interface.

1.6 Thread Safety

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

The EJB WebService implementations are technically mutable since the configured managers and resources are set after construction, but the container will not initialize the properties and resources 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 implementations can be safely accessed from multiple threads in EJB container.

The ***web service clients*** are stateless and there are no explicit data store concurrency issues that are in the scope of this component so they are thread-safe.

JPAUserMappingRetriever is thread-safe too, its entityManager is expected to be set once and the container ensures its thread safety too.

2. Environment Requirements

2.1 Environment

- Sun Java 1.5 is required for development and Java 1.5 and Java 1.6 for compilation
- RedHat Linux 7
- Solaris 7
- Informix 10
- EJB3
- J2EE
- Windows 200, 2003

2.2 TopCoder Software Components

- **Client Project Entities DAO 1.0:** its entities are used in this design.
- **Client Project Management 1.0:** its managers are used in this design.
- **Client Project Lookup Services 1.0:** its Roles interface is used in this design.
- **JBoss Login Module 2.0** – is used to perform user's authentications.
- **Logging Wrapper 2.0** – is used to perform the logging.
- **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 Configuration API Plugin 1.0 –** is used in the creation of ObjectFactory.
- **Object Factory 2.1:** This is used to instantiate configured objects as needed by this component.
- ***Base Exception 2.0*** *– it is not used because of some known issues with JBoss.*
- ***Search Builder 1.4*** *– it is not is used in this design. Only Client Project Lookup Services 1.0 performs lookups.*

2.3 Third Party Components

- None

3. Installation and Configuration

3.1 Package Name

- **com.topcoder.clients.webservices –** contains the web services interfaces, their local and

remote interfaces and their corresponding exceptions;

- **com.topcoder.clients.webservices.beans -** contains the web services implementations and their corresponding exceptions;
- **com.topcoder.clients.webservices.webservicesclients -** contains the web services clients and their corresponding exceptions.
- **com.topcoder.clients.webservices.usermapping –** contains UserMappingRetriever interface and the corresponding exceptions.
- **com.topcoder.clients.webservices.usermapping.impl -** contains UserMappingRetriever JPA implementations, needed entities and the corresponding exceptions.

3.2 Configuration Parameters

3.2.1 Resources

Configuration of resources needed in beans is realized using @Resource(name = resourceName). Here **xxx** means **company**, **client** and **project**.

Parameter	Description	Values
xxxManagerFile	Represents the manager configuration file name. ***Required***.	**String**. Can not be null or empty.
xxxManagerNamespace	Represents the manager configuration namespace. ***Required***.	**String**. Can not be null or empty.
userMappingRetrieverFile	Represents the user mapping retriever configuration file name. ***Required***.	**String**. Can not be null or empty.
userMappingRetriever Namespace	Represents the user mapping retriever configuration namespace. ***Required***.	**String**. Can not be null or empty.
logName	Represents the configuration log name. ***Optional***.	**String**. Can be null or empty.
aAdministratorRole	Represents the configuration administrator role. ***Optional***.	**String**. Can be null or empty.
clientAndProjectUserRole	Represents the configuration client and project user role. ***Optional***.	**String**. Can be null or empty.

Below is a sample configuration for the resources:

```
<env-entry>
       <description>The name of the log to use</description>
       <env-name>logName</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>log_Name</env-value>
</env-entry>

<env-entry>
       <description>The client manager file to use</description>
       <env-name>clientManagerFile</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>clientManagerFile</env-value>
</env-entry>

<env-entry>
       <description>The project manager file to use</description>
       <env-name>projectManagerFile</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>projectManagerFile</env-value>
```

```
</env-entry>

<env-entry>
       <description>The company manager file to use</description>
       <env-name>companyManagerFile</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>companyManagerFile</env-value>
</env-entry>

<env-entry>
       <description>The client manager namespace to use</description>
       <env-name>clientManagerNamespace</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>clientManagerNamespace</env-value>
</env-entry>

<env-entry>
       <description>The project manager namespace to use</description>
       <env-name>projectManagerNamespace</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>projectManagerNamespace</env-value>
</env-entry>

<env-entry>
       <description>The company manager namespace to use</description>
       <env-name>companyManagerNamespace</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>companyManagerNamespace</env-value>
</env-entry>

<env-entry>
       <description>The user mapping retriever file to use</description>
       <env-name>userMappingRetrieverFile</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>userMappingRetrieverFile</env-value>
</env-entry>

<env-entry>
       <description>The user mapping retriever namespace to use</description>
       <env-name>userMappingRetrieverNamespace</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>userMappingRetrieverNamespace</env-value>
</env-entry>


<env-entry>
       <description>The name of the administrator role</description>
       <env-name>administratorRole</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>Admin</env-value>
</env-entry>

<env-entry>
       <description>The name of the client and project user role</description>
       <env-name>clientAndProjectUserRole</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>Client and Projects User</env-value>
</env-entry>
```

3.2.2 Configuration properties inside the ConfigurationObject

Here **xxx** means **company_manager**, **client_manager, project_manager and user_mapping_retriever.**

Parameter	Description	Values
xxx_token	Represents the token used to get the needed Manager instance via object factory. ***Required.***	**String.** Can not be null or empty.

Below is a sample configuration:

- For **ProjectServiceBean**:

```
<CMConfig>

    <Config name="ProjectServiceBean">

        <Property name="project_manager_token">
            <Value>DAOProjectManager</Value>
        </Property>

       <Property name="user_mapping_retriever_token">
            <Value>JPAUserMappingRetriever</Value>
        </Property>

    </Config>
</CMConfig>
```

- For **ClientServiceBean**:

```
<CMConfig>

    <Config name="ClientServiceBean">

        <Property name="client_manager_token">
            <Value>DAOClientManager</Value>
        </Property>

       <Property name="user_mapping_retriever_token">
            <Value>JPAUserMappingRetriever</Value>
        </Property>

    </Config>
</CMConfig>
```

- For **CompanyServiceBean**:

```
<CMConfig>

    <Config name="CompanyServiceBean">

        <Property name="company_manager_token">
            <Value>DAOCompanyManager</Value>
        </Property>

    </Config>
</CMConfig>
```

3.3 Dependencies Configuration

- EJB 3.0 and TopCoder dependencies should be properly configured.

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

- 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 provided web services.

4.3.1 Demo for ProjectService

```
// we have the following WSDL document location:
String wsdlLocation =
"http://topcoder.com:8080/jaxws/clientProject/ProjectService/?wsdl";

// create a ProjectServiceClient client:
ProjectServiceClient projectServiceClient =
       new ProjectServiceClient (wsdlLocation);

// get ProjectService endpoint:
ProjectService projectService=
projectServiceClient.getProjectServicePort();


// we assume that there is already a client and a project status
// in the persistence:
Client client = …;
ProjectStatus projectStatus = …;
Project project;

project = new Project();
project.setName("project name");
project.setDescription("project's description");
project.setPOBoxNumber("111222333");

client.setName("client name");
client.setSalesTax(2223.5);

// create a project for the given client:
project= projectService.createProject(project, client);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// update a project:
project.setName("project's another name ");
project.setDescription("project's another description");
project.setPOBoxNumber("555222333");
client.setName("client's another name");
client.setSalesTax(1114.6);
project.setClient(client);

project = projectService.updateProject(project);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// set a project's status:
projectStatus.setName("project status name");
projectStatus.setDescription("project status description");
project = projectService.setProjectStatus(project, projectStatus);

// delete a project:
projectService.deleteProject(project);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.
```

4.3.2 Demo for ClientService

```
// we have the following WSDL document location:
String wsdlLocation =
"http://topcoder.com:8080/jaxws/clientProject/ClientService/?wsdl";

// create a ClientServiceClient client:
ClientServiceClient clientServiceClient =
      new ClientServiceClient (wsdlLocation);

// get ClientService endpoint:
ClientService clientService=
clientServiceClient.getClientServicePort();

// we assume that there is already a client status in the persistence:
ClientStatus clientStatus = …;
Client client;

client = new Client();
client.setName("client name");
client.setSalesTax(2223.5);

// create a client:
client = clientService.createClient(client);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// update a client:
client.setName("client's another name");
client.setSalesTax(1114.6);
client = clientService.updateClient(client);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// set a client's code name:
client = clientService.setClientCodeName(client, "client code name");
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.


// set a client's status:
clientStatus.setDescription("client status description");
client = clientService.setClientStatus(client, clientStatus);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// delete a client:
clientService.deleteClient(client);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.
```

4.3.3 Demo for CompanyService

```
// we have the following WSDL document location:
String wsdlLocation =
"http://topcoder.com:8080/jaxws/clientProject/CompanyService/?wsdl";

// create a CompanyServiceClient client:
CompanyServiceClient companyServiceClient =
      new CompanyServiceClient (wsdlLocation);

// get CompanyService endpoint:
CompanyService companyService=
companyServiceClient.getCompanyServicePort();
```

```
Company company;

company = new Client();
company.setName("company name");
company.setPasscode("12213");

// create a company:
company = companyService.createCompany(company);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// update a company:
company.setName("company's another name");
company.setPasscode("23324");
company = companyService.updateCompany(company);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// delete a company:
companyService.deleteCompany(company);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.
```

5. Future Enhancements

- If the Client and Projects Management component is updated, this component will be updated to reflect the changes.