Team Services 1.0 Component Specification

1. Design

This component implements all the business rules governing the process of building teams for competitions. It relies on a lower layer of components that manages teams, projects, resources and users. A team is a group of resources working for a project. It has basic info (a header), a set of filled or unfilled positions and a set of custom properties. A team can also be finalized. A finalized team cannot contain unfilled positions.

The main interface of this component is fine grained enough to provide a useful API to client applications, but coarse grained enough to offer transactional atomic services and allow the presentation layer to minimize the calls to this layer.

This design makes use of the Logging Wrapper to facilitate informational and debugging logging in the services implementation. Logging is not performed inside the bean implementations as there is no need to know such granular and trivial information. Logging is optional.

1.1 Design considerations

1.1.1 EJBs, ConfigManager, and Logging Wrapper

The EJB specification stipulates that File I/O is not allowed during the execution of the bean. At first glance this might mean that the use of the ConfigManager, and by extension Object Factory, could not be used because it performs property retrieval and storing using files. However, the restriction is only placed on the bean's lifetime, not the ConfigManager's. Therefore, as long as the ConfigManager does not perform I/O itself during the execution of the bean, or to be more accurately, that the thread performing a bean operation does not cause the ConfigManager to perform I/O, the use of ConfigManager to hold an in-memory library of properties is acceptable. Therefore, this design makes extensive use of ConfigManager and Object Factory, again, with the stipulation that the ConfigManager implementation does not perform I/O when this component is retrieving properties.

The issue of using Logging Wrapper is similar. The user of this component must ensure that logging is performed in a manner that does not violate EJB standards, such as not writing to a file or console.

1.2 Design Patterns

1.2.1 Strategy

TeamServicesImpl uses the various external managers and services in a transparent manner.

1.2.2 Façade

The **Façade** pattern can be said to be used in the TeamServices, as it provides streamlined access to managing teams and registrations that uses numerous other components.

1.3 Industry Standards

- EJB (specifically, to ensure this component is compatible with EJB restrictions, as defined in http://java.sun.com/blueprints/qanda/ejb_tier/restrictions.html).
- JavaBeans

1.4 Required Algorithms

1.4.1 Logging

This section is the central place to describe how logging is performed in lieu of stating this in each method in the ZUML.

All methods in TeamServicesImpl have access to a Log and should log in the following manner:

- Method entrance and exit at INFO level.
- All exceptions and errors at ERROR level. This includes illegal arguments.
- All calls to external TopCoder classes (just to managers and services classes, not beans) at DEBUG level. This includes before the call and after the call.
- Any additional logging is at the developer's discretion.

Note that if the TeamServicesImpl's log is null, no logging is to be performed.

1.4.2 Template XML Data Generation

Several operations require the generation of messages using the Document Generator component. Part of the task will be to create the required layered XML that will pass the data to the messages. This section will define the structure that each service method involved in messaging will adhere to.

The following sections will show the XML formats that the developer will assemble. Simple string concatenation will be sufficient. Note that all the information will be available for this in the service methods.

1.4.2.1 RemoveTeam service XML data format

This message will only require information about the team to be sent, so a portion of the header information pertaining to the removed team will be sent.

The following information will be passed: Team name, Project name, captain handle, and Team Description.

```
<DATA>
    <TEAM_NAME>name</TEAM_NAME>
    <TEAM_DESCRIPTION>description</TEAM_DESCRIPTION>
    <PROJECT_NAME>project name</PROJECT_NAME>
```

```
    <CAPTAIN_HANDLE>handle</CAPTAIN_HANDLE>
<DATA>
```

1.4.2.2 SendOffer service XML data format

This message will require the offer information. Basically, we will put together the data in the Offer object, with the positionId changed to position name, and the handles of the user instead of their IDs.

```
<DATA>
    <OFFER>
        <FROM>joe</FROM >
        <TO>jim</TO>
        <POSITION>futile</POSITION>
        <STATUS>offered</STATUS>
        <PERCENTAGE>25</PERCENTAGE>
        <MESSAGE>Please accept</MESSAGE>
        <REJECTION>Because I can</REJECTION>
    </OFFER>
<DATA>
```

1.4.2.3 AcceptOffer service XML data format

This message will require the offer information. Basically, we will put together the data in the Offer object, with the positionId changed to position name, and the handles of the user instead of their IDs.

```
<DATA>
    <OFFER>
        <FROM>joe</FROM >
        <TO>jim</TO>
        <POSITION>futile</POSITION>
        <STATUS>offered</STATUS>
        <PERCENTAGE>25</PERCENTAGE>
        <MESSAGE>Please accept</MESSAGE>
        <REJECTION>Because I can</REJECTION>
    </OFFER>
<DATA>
```

1.4.2.4 RejectOffer service XML data format

This message will require the offer information. Basically, we will put together the data in the Offer object, with the positionId changed to position name, and the handles of the user instead of their IDs.

```
<DATA>
    <OFFER>
        <FROM>joe</FROM >
        <TO>jim</TO>
        <POSITION>futile</POSITION>
        <STATUS>offered</STATUS>
        <PERCENTAGE>25</PERCENTAGE>
        <MESSAGE>Please accept</MESSAGE>
        <REJECTION>Because I can</REJECTION>
    </OFFER>
<DATA>
```

1.4.2.5 RemoveMember service XML data format

This message will provide details about the member being removed. This will include the position and team details for this resource.

```
<DATA>
```

```
    <HANDLE>removed</HANDLE>
    <POSITION_NAME>position name</POSITION_NAME>

    <TEAM_NAME>name</TEAM_NAME>
    <PROJECT_NAME>project name</PROJECT_NAME>
    <CAPTAIN_HANDLE>handle</CAPTAIN_HANDLE>
<DATA>
```

1.5 Component Class Overview

1.5.1 com.topcoder.registration.team.service

This package holds all the interfaces and constants in this component.

TeamServices
This represents the interface that defines all business methods for team services. The services include being able to manage teams and their positions as well as the resources that fill them. In fact, this service provides the ability to manage the offers for these positions to resources as well as sending notifications regarding those offers.

It has one implementation: TeamServicesImpl.

Thread Safety: There are no restrictions on thread safety in implementations.

ResourcePosition
The interface that defines the information about the position a resource occupies in a team, as well as providing the resource info itself. Extends java.io.Serializable so it can be used in a remote environment.

This interface follows java bean conventions for defining setters and getters for these properties. The implementations are to provide the implementations of these and at least an empty constructor.

It has one implementation: ResourcePositionImpl.

Thread Safety: There are no restrictions on thread safety in implementations.

OperationResult
The interface that defines the result of a team service operation. It defines a flag for success, and potential error messages if the team service operation was not successful. Extends java.io.Serializable so it can be used in a remote environment.

This interface follows java bean conventions for defining getters for these properties. The implementations are to provide the setters and at least an empty constructor.

It has one implementation: OperationResultImpl.

Thread Safety: There are no restrictions on thread safety in implementations.

1.5.2 *com.topcoder.registration.team.service.impl*

This package holds the implementations to all interfaces in the com.topcoder.registration.team.service package.

TeamServicesImpl
Full implementation of the TeamServices interface. This implementation makes use of a large array of components to accomplish its task of managing teams and offers: User Data Store Persistence component to retrieve full user information. It uses the Phase and Resource Management, Project Phases, Project Services, Team Management, OfferManagement, and Contact Member Service.

To provide a good view as the steps are progressing in each method, as well as full debug information for developers, the Logging Wrapper component is used in each method. To configure this component, the ConfigManager and ObjectFactory components are used.

Thread Safety: This class is immutable but operates on non thread safe objects, thus making it potentially non thread safe.

ResourcePositionImpl
Simple implementation of the ResourcePosition interface. Implements all methods in that interface, and provides the a default constructor if the user wants to set all values via the setters, and one full constructor to set these values in one go

Thread Safety: This class is mutable and not thread safe.

OperationResultImpl
Simple implementation of the OperationResult interface. Implements all methods in that interface, and adds the required setters. It provides the required default constructor if the user wants to set all values via the setters, and one full constructor to set these values in one go.

Thread Safety: This class is mutable and not thread safe.

1.6 Component Exception Definitions

This component defines two custom exceptions. Note that all are runtime exceptions.

TeamServiceConfigurationException
Extends BaseRuntimeException. Called by the TeamServicesImpl constructors if a configuration-related error occurs, such as a namespace not being recognized by ConfigManager, or missing required values, or errors while constructing the managers and services.

UnknownEntityException
Extends BaseRuntimeException. Called by all TeamServices methods that take a teamId, projectId, positionId, or resourceId when that ID is not found in the manager.

1.7 Thread Safety

Thread safety is mentioned as a required part of this component. The component is virtually thread-safe, and under expected conditions, it is effectively thread-safe.

Objects such as OperationResultImpl and ResourcePositionImpl are not thread-safe, and if had to be made thread-safe, its read/write operations would have to lock the fields they access.

Another aspect is the use of mutable, non-thread-safe objects in the TeamServicesImpl methods. This effectively renders TeamServicesImpl non-thread-safe and the only way to make it thread-safe is to ensure all objects it uses are thread-safe. However, it is expected that these bean instances will be used by multiple threads, so this is not an issue.

In terms of transactional control, this component is assumed to work in an EJB environment. As such, it takes no steps to manually roll back any steps if errors are encountered.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.4
- Compile target: Java 1.4

2.2 TopCoder Software Components

- Configuration Manager 2.1.5
    - Used for configuration of TeamServicesImpl
- Object Factory 2.0.1
    - Used to obtain instances of required objects in TeamServicesImpl. Some of the required objects are instances of PhaseManager, ResourceManager, etc…
- Base Exception 2.0
    - ToCoder standard for all custom exceptions.
- Logging Wrapper 2.0
    - Used for logging actions in TeamServicesImpl. A Log instance is obtained from LogManager.

- Document Generator 2.0
    - Provides the facility to generate messages.
- Project Management 1.1
    - Provides the classes and interfaces used for phase management: ProjectManager, Project, and ProjectCategory.
- Resource Management 1.1
    - Provides the classes and interfaces used for resource management: ResourceManager, Resource, and ResourceRole.
- Project Phases 2.0
    - Provides the phase model: Project and Phase.
- User Project Data Store 2.0
    - Provides the UserRetrieval manager used to retrieve ExternalUsers.
- Phase Management 1.1
    - Provides the interface used for phase management: PhaseManager.
- Project Services
    - Provides the ProjectServices and FullProjectData interfaces. This component is used for getting active project data.
- Offer Management
    - Provides offer management for dealing with offers for team positions to potential resources.
- Team Management 1.0
    - Provides team and position management via the TeamManager interface
- Contact Member Services
    - Provides the services to contact members about projects, teams and offers.

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

There are no third party components that need to be used directly by this component.

3. Installation and Configuration

3.1 Package Names

com.topcoder.registration.team.service
com.topcoder.registration.team.service.impl

3.2 Configuration Parameters

TeamServicesImpl

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required.	Example: “com.topcoder.specify”
userRetrievalKey	Key for the UserRetrieval to pass to ObjectFactory. Required	Valid key
resourceManagerKey	Key for the ResourceManager to pass to ObjectFactory. Required.	Valid key
projectServicesKey	Key for the ProjectServices to pass to ObjectFactory. Required.	Valid key
phaseManagerKey	Key for the PhaseManager to pass to ObjectFactory. Required	Valid key
projectManagerKey	Key for the ProjectManager to pass to ObjectFactory. Required	Valid key
teamManagerKey	Key for the TeamManager to pass to ObjectFactory. Required	Valid key
offerManagerKey	Key for the OfferManager to pass to ObjectFactory. Required	Valid key
contactMemberServiceKey	Key for the ContactMemberService to pass to ObjectFactory. Required	Valid key
loggerName	The name of the log to get from the LogManager. Optional	A valid name of the log. If not given, logging is not performed.
registrationPhaseId	The ID of the registration phase. Required	A non-negative long number.
freeAgentRoleId	The ID of a free agent resource role. Required	A non-negative long number.
teamCaptainRoleId	The ID of a team captain resource role. Required	A non-negative long number.
submitterRoleId	The ID of a submitter resource role. Required.	A non-negative long number.
removeTeamMessage TemplateName	The name of the template to use for generating messages for team removal. Required.	A valid template name in DocumentGenerator
sendOfferTemplateName	The name of the template to use for generating messages for offer sending. Required.	A valid template name in DocumentGenerator
acceptOfferTemplateName	The name of the template to use for generating messages for offer acceptance. Required	A valid template name in DocumentGenerator
rejectOfferTemplateName	The name of the template to use for generating messages for offer rejection. Required.	A valid template name in DocumentGenerator
removeMemberMessage TemplateName	The name of the template to use for generating messages for member removal. Required.	A valid template name in DocumentGenerator

3.3 Dependencies Configuration

3.3.1 TopCoder dependencies

The developer should refer to the component specification of the TopCoder components used in this component to configure them. Please see section 2.2 for a list of these components.

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

None

4.3 Demo

4.3.1 Setup

Much of the setup for any demo involves properly configuring all the components. This demo will not go into that depth, but it will show a sample configuration XML file.

Here we define the configuration parameters in ConfigManager:

```
<Config name="TestConfig">
   <Property name="specNamespace">
         <Value>SpecificationNamespace</Value>
   </Property>
   <Property name="userRetrievalKey">
         <Value>userRetrievalKey</Value>
   </Property>
   <Property name="resourceManagerKey">
         <Value>resourceManagerKey</Value>
   </Property>
   <Property name="projectServicesKey">
         <Value>projectServicesKey</Value>
   </Property>
   <Property name="phaseManagerKey">
         <Value>phaseManagerKey</Value>
   </Property>
   <Property name="projectManagerKey">
         <Value>projectManagerKey</Value>
   </Property>
   <Property name="teamManagerKey">
         <Value>teamManagerKey</Value>
   </Property>
   <Property name="offerManagerKey">
         <Value>offerManagerKey</Value>
   </Property>
   <Property name="contactMemberServiceKey">
         <Value>contactMemberServiceKey</Value>
   </Property>
   <Property name="loggerName">
         <Value>defaultLogger</Value>
```

```
    </Property>
    <Property name="registrationPhaseId">
          <Value>1</Value>
    </Property>
    <Property name="freeAgentRoleId">
          <Value>1</Value>
    </Property>
    <Property name="teamCaptainRoleId">
          <Value>2</Value>
    </Property>
    <Property name="submitterRoleId">
          <Value>3</Value>
    </Property>
    <Property name="removeTeamMessageTemplateName">
          <Value>removeTeamMessageTemplate</Value>
    </Property>
    <Property name="sendOfferTemplateName">
          <Value>sendOfferTemplate</Value>
    </Property>
    <Property name="acceptOfferTemplateName">
          <Value>acceptOfferTemplate</Value>
    </Property>
    <Property name="rejectOfferTemplateName">
          <Value>rejectOfferTemplate</Value>
    </Property>
    <Property name="removeMemberMessageTemplateName">
          <Value>removeMemberMessageTemplate</Value>
    </Property>
</Config>
```

For the demo, we will assume there is project with ID 1, and that there are several teams registered, including team ID 10. A pool of free agents may include resources with IDs 2-20.

The actual construction of the services class would be done in the following manner.

```
// Create TeamServicesImpl from configuration.
TeamServices service = new TeamServicesImpl(“TestConfig”);
```

4.3.2 *Usage*

A typical usage scenario involves the full use of the methods to manage teams and offers. We use the service instance and scenario setup from above. We will have a use with ID = 1 for all calls that use it. This demo will basically run through a typical team setup run.

```
// Create a team to project 1
TeamHeader teamHeader = // new team with project ID = 1, captain resource
#1, and 50 percent payment
OperationResult result = service.createOrUpdateTeam(teamHeader, 1);
// The result would indicate that the operation was successful. The team
is created with ID = 11.

// Remove a team 10 from project 1
OperationResult result = service.removeTeam(11, 1);
// Team with ID 10 is removed, and a message sent to the team members
```

```
// We will search for free agents in this project, and select some to add
to the positions.
Resource[] freeAgents = service.getFreeAgents(1);
// 19 resources would be returned.

// We will now add positions to the new team
Position position = // new position for resource # 2, 30 percent payment
result = service.createOrUpdatePosition(position, 11, 1);
position = // new position for resource # 3, 20 percent payment
result = service.createOrUpdatePosition(position, 11, 1);
// The results would indicate that the operations were successful. The
positions are created with ID = 101 and 102, respectively.

// We can retrieve a position and team we just created
position = service.getPosition(101);
team = service.getTeam(11);

// We can also get the resources in this team (11)
Resource[] resources = service.getTeamMembers(11);
// We would get the three resources on this project: IDs 1, 2, and 3

// If we want more details about the positions, we can use another
service method
ResourcePosition[] details = service.getTeamPositionsDetails(11);
// We would get the two positions and the resources for resource IDs 2
and 3

// Other bookkeeping service methods may include removing a position from
another team.
result = service.removePosition(40,1);
// position 40 would be removed

// Suppose we want to remove resource 3 from the team, and send offers to
other free agents. We begin by removing the resource.
result = service.removeMember(3,1);
// resource 3 is removed from position 102

// We then send an offer to resource 4
Offer newOffer = // offer to resource 4 to join team 11
result = service.sendOffer(offer);
// Offer assigned ID 1234 and sent to resource 4

// unfortunately, resource 4 will reject it
result = service.rejectOffer(1234, “Too Busy”, 4);

// we then try resource 5, and she accepts
result = service.acceptOffer(1235, 5);

// For other teams, we also check out offers
Offer[] offers = service.getOffers(6);

// We the proceed with finalizing the team 11, since all positions are
filled. We first validate
result = service.validateFinalization(11,1);
// The result is ok, so we proceed to finalize
result = service.finalizeTeam(11,1);
// Team 11 is now finalized

// At this time, we may also finalize other teams in this project. We
will just force them to finalize, if possible
result = service.forceFinalization(1,1);
// The result is ok, all teams in project 1 are finalized.
```

```
// This demo has provided a typical scenario for the usage of the
service.
```

5. Future Enhancements

Registration to non-team competitions will be added, along with new services like “Unregister to Project”