Submission Manager 1.2 Component Specification

Red for new texts, **blue** for updated texts.

1. Design

This component provides operations on contest like add CRUD operations on the submission, prize, review and submission payment; update submission status and placement; add and remove prizes from submission. Component runs as stateless EJB. This component is used Hibernate JPA implementation to work with persistence. It is used by Studio Service and can be used for the other services.

This component comprises a stateless session EJB for creating, storing, and loading instances of the Prize, Submission, SubmissionPayment, and SubmissionReview. The session EJB can be accessed via a remote or local interface. Logging can be added.

In version 1.2, three new methods are added into SubmissionManager interface and its EJB implementation SubmissionManagerBean, to support get milestone submissions, get final submissions and set submission milestone prize. Besides, old document and diagrams are updated to get consistent with the latest implementation(source code).

1.1 Design Patterns

Strategy: SubmissionManagerBean is a concrete implementation of SubmissionManager, it can be plugged into appropriate context(The context is not defined in this component.)
Version 1.2 doesn't bring new design patterns.

1.2 Industry Standards

- EJB 3.0
- JPA 1.0

This part is not changed in version 1.2.

1.3 Required Algorithms

Version 1.2 doesn't bring any new substantial algorithms, new methods added follow the similar process with old methods. Noted that the description of logging logic is updated here to get consistent with the current implementation.

1.3.1 Handling collisions when updating submission results

The approach to updating ranks in a submission must incorporate the scenario where the update involves a rank already claimed by another submission.

For example, suppose that after appeal responses stage, ranks are assigned, but there are pending re-appeals that eventually bump a submission to a new rank. In such a case, the other ranked submissions may have to be automatically changed.

More formally, suppose we have the following rankings after appeal response phase:

submission1: rank 1
submission2: rank 2
submission3: rank 3
submission4: rank 4

Suppose that after a re-appeal, submission3 is awarded first place. The update algorithm must then bump the other submissions down, so we end up in the following new order:

submission3: rank 1
submission1: rank 2
submission2: rank 3
submission4: rank 4

The algorithm will be as follows (This is a simple recursive step):
1. For promotion, determine if there is any existing submission that currently occupies the rank of the submission being updated. If yes, repeat this step with this existing submission as being the updated one, but its rank bumped by 1.
2. For demotion, the same algorithm applied, just decrease rank instead of bump.

So let's see how this plays out, step-by-step, in our example. We start with, as above, with submission3 now having rank 1.

We apply the algorithm and find that submission1 currently occupies that rank. We thus save submission3 with the new rank (and prizes for this rank), and assign submission1 a rank of 2.

submission3: rank 1, submission1: rank 2
submission2: rank 2
submission4: rank 4

We now repeat the process for submission1. We quickly see that there is submission2 that also has this rank. So we save submission1 with rank 2 (and again, the prizes for that rank), and bump submission2's rank to 3.

submission3: rank 1
submission1: rank 2
submission2: rank 3
submission4: rank 4

Now we repeat the process for submission2. We quickly see that there is no conflicting submission, so we have arrived at the end of the process. We simply save submission2 with rank 3 (and the appropriate prizes for that rank), and we are done.

1.3.2 Logging standards

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

Logging will be performed in every business operation in the EJB if the logger is configured for it. There will be no logging in any constructor or initialization method anywhere.

Logging should be performed in the following common manner:
- All method entrance and exit will be logged with DEBUG level.
    - Entrance format: [Entering method {*className.methodName}]*
    - Exit format: [Exiting method {*className.methodName}*]. Only do this if there are no exceptions.
- All exceptions will be logged at ERROR level. When logging exceptions, log the inner exceptions as well
    - Format: Simply log the text of exception and inner exceptions: [Error in

method {*className.methodName}*: Details {*error details*}]

Any other logging is at the developer's discretion. The developer is also free to improve on the above template. Such logging should be at the DEBUG level.

1.3.3 EJB transactions and exceptions

The current EJB framework provides for two means of handling application exceptions (of which all non-runtime custom exceptions are application exceptions).

- We can put into the code of each method handling of each exception to roll back be transaction with the use of the session context's setRollbackOnly(true).
- We can configure in the XML or via annotations the exceptions to cause a rollback.

the second option (with either the XML or annotation approach) is chosen by this implementation, by adding @ApplicationException(rollback = true) for all custom exception.

1.4 Component Class Overview

SubmissionManager
The business interface that defines methods for managing a submission, prize, submission payment, and submission review. In general, but not always, it has methods to create, update, delete, get, and list these entities, as well as some methods to perform more operations. These include getting submissions for a contest or member, updating a submission status or result, adding or removing a prize from a submission, and getting and deleting submission reviews for a submission and/or a reviewer.

In version 1.2, three methods are added to support get milestone submissions, get final submissions and set submission milestone prize. Besides, the document within TCUML is updated to get consistent with the current implementation. Especially, several existing methods are totally missing in the original diagram, they are added in this version.

SubmissionManagerLocal
The local EJB interface that simply extends the SubmissionManager interface, with no additional facilities. This interface should be marked with @Local annotation representing it's a remote interface.

There is no change for this interface in version 1.2.

SubmissionManagerRemote
The remote EJB interface that simply extends the SubmissionManager interface, with no additional facilities. This interface should be marked with @Remote annotation representing it's a remote interface.

There is no change for this interface in version 1.2.

SubmissionManagerBean
The stateless session bean that performs the CRUDE specified by the SubmissionManagerRemote and SubmissionManagerLocal interfaces. It uses JPA perform all operations and the Log for logging.

This interface should be marked with @Stateless annotation representing it's a stateless session bean. Also, it must have the @TransactionAttribute(REQUIRED) annotation to indicate all operations require transactional control. Finally, it will add the annotations @DeclareRoles({ "Cockpit User", "Cockpit Administrator" }) and @RolesAllowed("Cockpit Administrator").

In version 1.2, three methods are added to support get milestone submissions, get final submissions and set submission milestone prize. Besides, the document within TCUML is updated to get consistent with the current implementation. Especially, several existing methods are totally missing in the original diagram, they are added in this version.

1.5 Component Exception Definitions

This component defines four custom exceptions.

In version 1.2 no new custom exceptions are brought and old custom exceptions are used as they were. Their documents in TCUML are updated to get consistent with the current implementations.

SubmissionManagementException
Extends BaseCriticalException. It is thrown by all CRUDE methods if there is an error during an operation. New methods added in version 1.2 can also throw this exception.

EntityNotFoundException
Extends SubmissionManagementException. It is thrown by the updateXXX and removeXXX methods if the given entity is not found in persistence. The new method added in version 1.2 to set submission milestone prize can also throw this exception.

EntityExistsException
Extends SubmissionManagementException. It is thrown by some of the addXXX methods if the given entity already exists in persistence. No new methods added in version 1.2 throw this exception.

SubmissionManagementConfigurationException
Extends BaseRuntimeException. Constitutes a general configuration exception that all implementations that require such configuration can throw it something goes wrong during that configuration. Called by the SubmissionManagerBean's *initialize* method if the configuration is not available. No new methods added in version 1.2 throw this exception.

NumberOfSubmissionsExceededException
Extends SubmissionManagementException. It is thrown by the setSubmissionMilestonePrize method if we've already reached the maximum number of submissions receiving milestone prizes for that contest.

InconsistentContestsException
Extends SubmissionManagementException. It is thrown by the setSubmissionMilestonePrize method if the contest the submission belongs to is not one of the contests set of the milestone prize.

1.6 Thread Safety

This component is effectively thread-safe. The SubmissionManagerBean is technically mutable since its session context, unit name, and logger properties are set after construction, but the container will not call the initialize method more than once for the session bean. The SubmissionManagerBean also operates on non-thread-safe entities, thus rendering it non-thread-safe. This is not an issue with the remote interface, since the bean will be called via the remote interface with an effective copy of the entities, the entity will not be shared by multiple threads. Therefore, the container makes the session bean effectively thread-safe in this case. The local interface is called in the same JVM without the overhead of a distributed object protocol, so entities will in fact be passed by reference, thus making it possible that multiple client threads, if they have access to the

same entity, to operate on it while the session bean works on it. However, the persistence container makes its own local copy of the entity instance per transaction basis, so we are protected from the issues of concurrent access to the entity.

All methods (create, update, and delete) are transitionally managed by the container. Transaction scope is Required.

There are no explicit data store concurrency issues that are in the scope of this component.

In version 1.2, all new methods added don't break the thread safety approach taken in the current implementation, the analysis for the old version's thread safety still applies in the new version.

2. Environment Requirements

2.1 Environment

- Java 1.5
- EJB 3.0
- Hibernate 3.2 or Higher
- JPA 1.0

  This part is not changed in version 1.2.

2.2 TopCoder Software Components

In version 1.2, it will use the updated version(1.2) of Contest and Submission Entities component.

- Base Exception 2.0
  - TopCoder standard for all custom exceptions.
- Contest and Submission Entities 1.2
  - Provides the entities used by this component.
- Logging Wrapper 2.0
  - Used for logging operations in the SubmissionManagerBean.

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.
This part is not changed in version 1.2.

3. Installation and Configuration

3.1 Package Names

com.topcoder.service.studio.submission
This part is not changed in version 1.2.

3.2 Configuration Parameters

Configuration occurs from JNDI ENC in the initialize method of the SubmissionManagerBean. The “unitName” must be provided as an environment entry, and it must point to a valid EntityManager. The “logger” entry is optional.
This part is not changed in version 1.2.

3.3 Dependencies Configuration

None
This part is not changed in version 1.2.

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.

This part is not changed in version 1.2.

4.2 Required steps to use the component

The SubmissionManager interfaces and bean should be deployed in an EJB 3.0 container.
This part is not changed in version 1.2.

4.3 Demo

This demo will show typical use of the management mechanism. This demo will mostly show how to use the remote interface to manage a Prize and Submission. The use of the local interface, and the management of the other entities (such as payments and reviews), are done in the same manner, and will not be shown.

Here’s the relevant excerpt from as possible deployment descriptor of the SubmissionManagerRemote bean and interfaces:

```
<enterprise-beans>
  <session>
    <ejb-name>SubmissionManager</ejb-name>
      <ejb-class>
com.topcoder.service.studio.submission.SubmissionManagerBean
      </ejb-class>
    <session-type>stateless</session-type>
    <transaction-type>Container</transaction-type>
    <resource-ref>
       <!-- This could define the actual DB connection info-->
    </resource-ref>
    <env-entry>
       <env-entry-name>unitName</env-entry-name>
       <env-entry-type>java.lang.String</env-entry-type>
       <env-entry-value>targetDB</env-entry-value>
    </env-entry>
    <env-entry>
```

```
            <env-entry-name>logger</env-entry-name>
            <env-entry-type>java.lang.String</env-entry-type>
            <env-entry-value>myLogger</env-entry-value>
        </env-entry>
    </session>
</enterprise-beans>
```

To obtain the SubmissionManagerRemote, the following typical JNDI code will be used:

```
// Get hold off remote interface
Properties env = new Properties();
env.setProperty(Context.SECURITY_PRINCIPAL, "admin");
env.setProperty(Context.SECURITY_CREDENTIALS, "password");
env.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.security.jndi.JndiLoginInitialContextFactory");
InitialContext ctx = new InitialContext(env);

manager = (SubmissionManager) ctx.lookup("remote/SubmissionManagerBean");
```

Then we can manage a Prize

```
            // create prize type
            PrizeType type = new PrizeType();
            type.setPrizeTypeId(1L);
            // Create a prize and fill it
            Prize prize = new Prize();
            prize.setPlace(1);
            prize.setAmount(900.00);
            prize.setCreateDate(new Date());
            prize.setType(type);
            // and other field are omitted for clarity
            Prize createdPrize = manager.addPrize(prize);
            // Checking the ID of the prize, it would be filled. We assume ID=11
            Long prizeId = createdPrize.getPrizeId();

            // At a later point, we can retrieve and update the prize to bump the
            // prize to half.
            Prize retrievedPrize = manager.getPrize(prizeId);
            prize.setPlace(2);
            prize.setAmount(450.00);
            manager.updatePrize(retrievedPrize);

            // We can remove the prize if it no longer meets our needs.
            manager.removePrize(retrievedPrize.getPrizeId());

            // This shows basic CRUD operations for a simple entity. We can proceed
            // to more involved examples of managing a submission:

            // Suppose we have 4 submissions for a given contest (contestId=1).
            // The relevant fields are to be denoted as:
            // submission 1: submissionId=1, rank=1
            // submission 2: submissionId=2, rank=2
            // submission 3: submissionId=3, rank=3
            // submission 4: submissionId=4, rank=4

            // If we wanted to retrieve and update some properties of a submission,
```

```
// we would do the following:
Submission submission = manager.getSubmission(2);
submission.setHeight(2);

manager.updateSubmission(submission);

// If we wanted to retrieve the submissions, without the actual files, we
// would do the following:
List<Submission> submissions = manager.getSubmissionsForContest(1, false);
// This would return a list of 4 submissions shown above

// Suppose that after additional appeals, the rankings were changed, and
// submission 3 was promoted to rank 1. The submission result would be
// updated:
Submission submission3 = manager.getSubmission(3);
submission.setRank(1);
manager.updateSubmissionResult(submission3);
// This would result in the rankings to be recalculated to accommodate
// the change in ranks. The submissions for the given contest would be
// adjusted as follows:
// submission 3: submissionId=3, rank=1
// submission 1: submissionId=1, rank=2
// submission 2: submissionId=2, rank=3
// submission 4: submissionId=4, rank=4
```

The following contents explain how the new methods added in version 1.2 work.

Suppose we have the following submissions stored currently:

Submission ID	ID of the contest it belongs to	Whether Deleted	Submission Type
1	1	Y	milestone_submission_type
2	1	N	milestone_submission_type
3	1	N	final_submission_type
4	1	N	final_submission_type
5	1	N	milestone_submission_type
6	2	N	milestone_submission_type

```
// Get milestone submissions under contest 1.
List<Submission> submissionList1 = manager.getMilestoneSubmissionsForContest(1);
/* submissionList1 will contain 2 elements, for submission 2 and 5. Submission 1 not
included since it has been deleted, submission 6 belongs to another contest, submission
3 and 4 are final submissions rather than milestone submissions. */

// Get final submissions under contest 1.
List<Submission> submissionList2 = manager.getFinalSubmissionsForContest(1);
/* submissionList2 will contain 2 elements, for submission 3 and 4. Submission 1 not
included since it has been deleted, submission 6 belongs to another contest, submission
2 and 5 are milestone submissions rather than final submissions. */
```

Suppose we have the following milestone prize stored:

Milestone prize ID	maximumNumberOfSubmissionsPerContest	Contests set(IDs) associated with the prize	Submissions set(IDs) already associated with the prize
1	3	{1,3}	{2}

// The following method call will not succeed:
manager.setSubmissionMilestonePrize(2, 1); /* Submission 2 already associated with milestone prize 1. */
manager.setSubmissionMilestonePrize(6, 1); /* Submission 6 belongs to contest 2, that contest doesn't exist in the contests set of milestone prize 1. */
manager.setSubmissionMilestonePrize(1, 1); /* Submission 1 is already deleted and can't be associated with the milestone prize */
manager.setSubmissionMilestonePrize(3, 1); /* Submission 3 is the final submission rather than milestone submission */
manager.setSubmissionMilestonePrize(7, 1); /* Submission 7 doesn't exist */
manager.setSubmissionMilestonePrize(5, 2); /* Milestone 2 doesn't exist */

// The only method call can succeed is:
manager.setSubmissionMilestonePrize(5, 1); /* That will add submission 5 into the submissions set of milestone prize 1 and save the changes. */

/* Please note, if the maximumNumberOfSubmissionsPerContest of milestone prize 1 is 1 rather than 3, even manager.setSubmissionMilestonePrize(5, 1) won't succeed since there is already 1 submission(id = 2) set to that milestone prize, no any more submissions can be associated if at most 1 submission can get that milestone prize. */

5. Future Enhancements

None
This part is not changed in version 1.2.