Registration Validation 1.0 Requirements Specification

1. Scope

1.1 Overview

This component provides an implementation of a validation plug-in for the Registration Services component. This plug-in performs all validation needed to make sure a registration to a will succeed if no runtime problems occur (such as a database crash).
Also it will be modeled in such a way that each piece of validation could be removed by configuration or new pieces added.

1.2 Logic Requirements

1.2.1 Interfaces

Validators must implement the RegistrationValidation interface defined in the Registration Services component.

1.2.2 Simple Validators

A set of simple validators will be implemented. These validators will be the building blocks for creating composite validators for an entire registration. These simple validators may as well be composed by other validators.
The following are the required simple validators.

1.2.2.1 Member not currently registered in a given role for the project

1.2.2.2 Member not currently registered as a Team Captain with assigned Team Members for this project.

1.2.2.3 Member has not exceeded the configured maximum number of current project registrations allowed.

1.2.2.4 Member is not currently a Team Member for this project.

1.2.2.5 Member is not currently barred from registering in new competitions.

1.2.2.6 Project is of a given type (and/or) a given category.

1.2.2.7 Project is currently in a given phase

1.2.2.8 Member has at least a given rating for a given rating type

1.2.2.9 Member has at least a given reliability for a given rating type

1.2.2.10 Member has at least a given volatility for a given rating type

1.2.2.11 Member has at least a given number of ratings for a given rating type

1.2.3 Custom validators

If a new custom validator is required, this component must allow the user to implement it and use it exactly as it was one of the Simple Validators.

1.2.4 Composite validations

This component must provide a way to create composite validations from configuration. Both conjunction (AND) and disjunction (OR) composites must be provided.
The conjunction must not continue performing validations after one has failed, except if otherwise is specified by configuration.

1.2.5 Validation negation

This component must provide a way to use a validator the inverse way. For example, for using 1.2.2.1 to validate the user is actually registered in a given role.
The result message must be optionally configurable.

1.2.6 Conditional validations

In order to provide validations for a wide range of registrations, this component must provide a way to configure which validations are to be performed based on the following information.

- Project Type
- Project Category
- Registering Resource Role
- Project Id

1.2.7 Examples

The following are examples of validation scenarios that should be possible to resolve through configuration of this component.

1.2.7.1 Team Captain and Free Agent

- All registering members must not be barred (1.2.2.5).
- If the project is of category “Assembly” (category Id=XX), then all registering members must not be currently a Team Member for this project (1.2.2.4).
- Members registering as Team Captain must not be currently a Team Captain for this project (1.2.2.1).
- Members registering as Free Agents must not be currently a Team Captain with assigned Team (1.2.2.2)

1.2.7.2 Tournaments

In case there’s a tournament going on, a new custom validator may be developed and added to the above configuration with the following rule:
If the project is of type “Contest” (type Id=XX), then all registering members must be registered to the tournament.

1.2.8 Return messages

The result of the each validation must be configurable as a compound message. For example, the message configuration for the 1.2.2.1 validator may look like

> “The member {member} is already registered to project {project} with role {role}.”

And a possible resulting message would be

> “The member ImMember is already registered to project Registration Validation 1.0 with role Team Captain”

The tags (or variables) available to use for a compound message will change on each Validator. It’s the designer’s responsibility to define an appropriate set of variables. As a guideline, at least the actual parameters received as the validation call arguments and the specific validator’s configurable values must be available.

1.2.9 Authentication

No authentication should be performed in this component. Authentication will be done in an upper layer.

1.2.10 Return values of Array type

For all methods returning arrays, in case there are no objects to fill the array it must return an empty one, never a null.

1.2.11 Access to the data model

This component uses existing lower level Entity Management components to execute the requests whenever necessary. It's required not to use the Data Access Objects directly since low level validations are performed in Entity Management layer.
The Entity Management layer comprises the following components:

- **Team Management**: Under development. Available in the catalog.
- **Resource Management**: Generic component. Available in the catalog.
- **Project Phases**: Generic component. Available in the catalog.
- **Project Management**: Generic component. Available in the catalog.
- **User Project Data Store**: Custom component. Provided

It's also allowed to use services form the Services Layer if necessary. The components in this layer are:

- Project Services
- Team Services
- Registration Services

The actual implementation of each Manager/Service must be pluggable.

1.2.12 Exception Management

If an exception occurs retrieving data from the dependent components and a validation cannot be performed, a RuntimeException subclass should be thrown.

1.2.13 Logging

This component must produce the appropriate logging information for tracing / debugging / production support.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

This component will be used by the Registration Services component for dynamically validating the registrations to competitions.

1.5 Future Component Direction

More validators may be added. Also conditionals based on other properties.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

2.1.2.1 Interfaces defined

This component must define and implement the interfaces depicted in the Component Interface Diagram inside the com.topcoder.registration.validation package.

2.1.2.2 Fields in CID interfaces

Some interfaces in the CID have member variables. The meaning of those members is that the actual Java interface will have getters and setters for those fields.

2.1.2.3 Under development interfaces

For using interfaces yet under development you must refer to the Component Interface Diagram provided and the following guidelines.

2.1.2.3.1 Team Manager

All TeamManager methods will throw specific checked exceptions on any business related problem. These exceptions are all subclasses of TeamManagerException, subclass of the com.topcoder.util.errorhandling.BaseException.
If the calling validator semantics requires it, this kind of problems may be added to the OperationResult objects.
Runtime problems like component misuse, invalid arguments or database connection must be thrown as unchecked exceptions.

2.1.2.3.2 Project Services

The Project Services will only throw RuntimeExceptions.

2.1.3 Environment Requirements

- Development language: Java1.4
- Compile target: Java1.4

2.1.4 Package Structure,

com.topcoder.registration.validation – Main package

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- The actual implementation to be used of the following components must be configurable (if applicable).
  Team Management
  Resource Management
  Project Phases
  Project Management
  Project Services
  Registration Services

- The whole validation strategy will be configurable. Each of the required validator has its own configuration requirements.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None.

3.2.2 TopCoder Software Component Dependencies:

Configuration Manager
Team Management
Resource Management
Project Phases
Project Management
User Project Data Store
Project Services
Registration Services
Logging Wrapper

3.2.3 Third Party Component, Library, or Product Dependencies:

None

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Windows 2003

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.3.1 EJB compliant

This component will be used from EJB objects, thus it must comply with EJB development restrictions (http://java.sun.com/blueprints/qanda/ejb_tier/restrictions.html).

3.3.2 Component Scalability

The component needs to be scalable. Running multiple instances in the same JVM or in multiple JVM's concurrently should not cause any problem.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.