Software Documentation : Java Custom Review Assignment

This page last changed on Nov 28, 2012 by volodymyrk.

1. Scope

1.1 Overview

TopCoder is improving its review process by collecting the reviewers' performance stats and modifying the review signup process so that the best performing reviewers have higher chances to get a review position. The first part of the system upgrade that added the reviewer rating into the Online Review application has been already implemented, see the details here. The next step is to modify the review signup process: instead of the "first come, first served" approach when members assign themselves to review positions they will *apply* for review positions and the system will select the best applicants for reviewing that project.

This component makes use of the API defined by the Project Management, Phase Management, Resource Management and Review Application Management components to regularly monitor pending review applications for active projects and assign applicants to review positions. The projects will be examined periodically.

1.2 Logic Requirements

1.2.1 Command Line Interface

The component will be a standalone Java utility that will provide a command line interface. Designer should use the TopCoder Command Line Utility component.

The following command line arguments will be supported:

1. Tracking Interval. Number of seconds between Review Assignment job executions.
2. Background flag. Indicates whether the tool is going to run in a background thread.
3. "Guard file" filepath. A path to the file to signal to the tool to stop the job.

1.2.2 Review Assignment job execution

Review Assignment will examine the projects at a configurable time interval. Designer should use the TopCoder Job Scheduling component.

Every job execution will do the following:

1. Get all auction categories (see the ReviewAuctionManager#getAuctionCategories() method).
2. For every auction category:
    a. Get the list of all open auctions for that category (see ReviewAuctionManager#searchOpenAuctions() method)
    b. For every open auction:
        i. Check if the assignment date has been already reached. If the assignment date is still in the future, skip this auction.
        ii. Get the list of all pending review applications for the auction (see ReviewApplicationManagerImpl#searchApplications() method and search filters from ReviewApplicationFilterBuilder). It will only include applications in the "Pending" status.
        iii. If there are no pending review applications for the auction, skip this auction.
        iv. Call a pluggable strategy to assign review applicants to the review application roles (see section 1.2.3). The input to that is the review auction and the list of pending review applications. The result is the map from review applications to review application roles.
        v. For every review application that was assigned a review application role:
            i. Update the review application status to "Approved".
            ii. Add the member to the Online Review project (see section 1.2.4).
            iii. Send a notification email (see section 1.2.5.1).
        vi. For every review application that was not assigned a review application role, update the review application status to "Rejected".

vii. For every distinct applicant (i.e. ReviewApplication#userId) that was not assigned to any review application role send a notification email (see section 1.2.5.2).
viii. Send a notification email to all PMs/copilots (see section 1.2.5.3).

1.2.3 Review Assignment strategy

The component will define a strategy interface to select best reviewers for a review auction (from those who have applied for it). The input will be the ReviewAuction instance and a list of ReviewApplication objects. The strategy will consider each review application and will either approve it by assigning it to one of the open review application roles or will reject it. The result of the strategy therefore will be a map from ReviewApplication to ReviewApplicationRole, where a null value means the review application is rejected.

The implementation of the strategy will be configurable, i.e. it will be possible to specify the implementation class name in the config file.

The component will also provide an implementation of the strategy that will select best reviewers using the following algorithm.

1. Group all review applications by user ID. One user can have more than one pending application for the same auction if they are for different application roles within that auction (however at most one of his applications will be approved since user can not have multiple roles within the same auction).
2. For all distinct applicants get their reviewer rating for this project category by running this SQL:

   ```
   SELECT rf.reviewer_user_id, AVG(rf.score) AS rating, COUNT(*) AS count
   FROM review_feedback rf
   INNER JOIN project p ON p.project_id = rf.project_id
   INNER JOIN project ref_p ON ref_p.project_id = PROJECT_ID AND ref_p.project_category_id =
   p.project_category_id
   WHERE rf.reviewer_user_id IN (LIST_OF_USER_IDs)
   GROUP BY rf.reviewer_user_id
   ```

   where PROJECT_ID is the value of ReviewAuction#projectId and LIST_OF_USER_IDs is the list of distinct ReviewApplication#userId values (see step #1).
   For those applicants who are not in the result set and those for which the count value is less than 4, assume the default rating value of 1.0
3. Solve the assignment problem. The task is to assign each applicant to at most one open review application role. Review application roles for the auction are defined in the ReviewAuction#auctionType.applicationRoles list. The number of open positions for each application role is defined in ReviewAuction#openPositions list, it defines the maximum number of applicants that can be assigned to each role. The task is to assign as many applicants as possible (thus fill as many open positions as possible) while selecting the applicants with highest reviewer rating (see step #2). This problem can be reduced to solving a maximum matchings in bipartite graphs if the open positions of the same review application role are considered as different graph nodes.
   The research of the algorithm is the designer's responsibility, but it should be noted that the total number of open review positions will typically be 3 or less and the total number of distinct applicants will usually be less than 10, so there's no need for a complex asymptotically optimal solution (which can be slower for small cases). It just needs to work fast for the small cases.

1.2.4 Adding reviewers to Online Review projects

When a review application is approved two things will happen: the member will be added to the project and some project phases will be extended if necessary.

1.2.4.1 Adding members to projects

When members are added to projects they are added as "resources". A resource has a role (e.g. "Primary Screener", "Accuracy Reviewer" etc.) and a number of additional properties. Resource Management component is used to add project resources. A member can have multiple resources in the same project.

A review *application role* can consist of multiple *resource roles* (e.g. "Primary Reviewer" application role consists of four resource roles: "Primary Screener", "Reviewer", "Aggregator", "Final Reviewer"). The list of resource roles for each application role is in the ReviewApplicationRole#resourceRoles list.

Here are the steps to add a member to the Online Review project for a specific review application role:

1. For each ReviewApplicationResourceRole in the ReviewApplicationRole#resourceRoles list:
   a. Get the ResourceRole instance by its ID value (ReviewApplicationResourceRole#resourceRoleId). This is done by calling the ResourceManager#getAllResourceRoles() method and creating a map from resource role ID to ResourceRole instances (see performance considerations from section 1.2.7).
   b. Create a new Resource instance:
      i. Set Resource#project to the ReviewAuction#projectId value.
      ii. Set Resource#resourceRole to the ResourceRole found earlier.
      iii. If the ResourceRole#phaseType is not null, find the Phase instance (see the PhaseManager#getPhases() method) whose Phase#phaseType.id value equals to ResourceRole#phaseType. Set Resource#phase to the Phase#id value.
      iv. Set the "External Reference ID" resource property to ReviewApplication#userId.
      v. Set the "Handle" resource property to the member's handle. Member's handle can be found by calling UserRetrieval#retrieveUser() method.
      vi. Set the "Registration Date" resource property to the current date and time. Datetime format should be "MM.dd.yyyy hh:mm a", locale US.
      vii. Set the "Payment" resource property to "0".
      viii. Set the "Payment Status" resource property to "No".
   c. Save the new Resource instance by calling ResourceManage#updateResource() method.
   d. Get the forum category ID associated with the project. This can be done by calling ProjectManager#getProject() method and then Project#getProperty("Developer Forum ID")
   e. If the forum category ID property is not null and not empty, call the Forums EJB to grant the member forum access: forumsBean.assignRole(userId, "Software_Users_" + FORUM_CATEGORY_ID);

1.2.4.2 Extending project phases

By the TopCoder rules, when a reviewer registers during the Review phase the phase needs to be extended to give the new reviewer at least 48 hours for the review. Similarly, if the primary reviewer registers during the Screening phase the phase needs to be extended to give him at least 6 hours.

For now the rules will be as follows:

1. If a resource has been added with ResourceRole#phaseType value equals to 3 (which corresponds to the Screening phase), and there's an open phase with that phase type ID, it needs to be extended to at least 6 hours from now.
2. If a resource has been added with ResourceRole#phaseType value equals to 4 (which corresponds to the Review phase), and there's an open phase with that phase type ID, it needs to be extended to at least 48 hours from now.
3. If a resource has been added with ResourceRole#phaseType value equals to 14 (which corresponds to the Specification Review phase), and there's an open phase with that phase type ID, it needs to be extended to at least 2 hours from now.

The actual phase type IDs and hours will configurable (don't hard-code the rules above, they were just an example).

Here are the steps to extend the phases:

1. Get all project phases by calling PhaseManager#getPhases() method.
2. For each project phase:
   a. Check if the phase is currently open (i.e. Phase#phaseStatus#id equals to PhaseStatus#OPEN#id) and if it needs to be extended according to the rules outlined above. If the phase is not open or doesn't need to be extended then skip to the next phase, otherwise proceed assuming the phase needs to be extended to at least X hours from now.
   b. Calculate the new value of the Phase#length field. The new value should be the duration (in milliseconds) from Phase#actualStartDate to X hours past the current time.
   c. Check the current Phase#length value. If if is larger then the calculated value then the phase doesn't need to be extended so skip to the next phase, otherwise update the Phase#length field.
3. Recalculate the start and end dates of all project phases by doing this:

```
for (int i = 0; i < allPhases.length; ++i) {
Phase phase = allPhases[i];
phase.setScheduledStartDate(phase.calcStartDate());
phase.setScheduledEndDate(phase.calcEndDate());
}
```
4. Persist the changes by calling PhaseManager#updatePhases() method.

1.2.5 Notification emails

Each time the tool assigns reviewers to a project it sends three types of notification emails:

1. Notification email to the members who were assigned to a review application role (i.e. those who have one Approved review application).
2. Notification email to the members who were not assigned to any review application role (i.e. those whose all review applications are Rejected).
3. Notification email to all PMs/copilots for that project.

All emails will be sent in the HTML format using the Email Engine and Document Generator components. The email templates, email subjects and sender's email addresses will be configurable for every notification type.

The member's email address can be found by calling the UserRetrieval#retrieveUser() method.

The designer is responsible to provide sample email templates demonstrating all supported fields for every notification type.

1.2.5.1 Notification emails to approved applicants

The content of the email will be created based on a configurable email template with the following fields:

1. Project Name
2. Project Version
3. Project ID
4. Member handle
5. Review application role name (i.e. ReviewApplicationRole#name value).

Project Name and Project Version can be found by calling ProjectManager#getProject() method and then Project#getProperty("Project Name") and Project#getProperty("Project Version").

1.2.5.2 Notification emails to rejected applicants

The content of the email will be created based on a configurable email template with the following fields:

1. Project Name
2. Project Version
3. Project ID
4. Member handle

1.2.5.3 Notification emails to PMs/copilots

The email will be sent only if at least one review application was approved (it may be possible that all applications are rejected, depending on the future implementations of the Review Assignment strategy).

To get the list of users to send the email notification to, retrieve the list of resources for that project. The list of the resource roles to be retrieved will be configurable (and will typically be Manager and Copilot). See the ResourceManager#searchResources() method.

The content of the email will be created based on a configurable email template with the following fields:

1. Project Name
2. Project Version
3. Project ID
4. For each approved review application:
    a. Member's handle.

b. The name of the review application role that was assigned to that member (i.e. ReviewApplicationRole#name value).

Loop construct from the Document Generator component should be used to handle the list of applicants.

1.2.6 Thread-Safety

The component is not explicitly required to be thread-safe since it will be used as a standalone Java application (if otherwise is not required by the Job Scheduling component).

However special care should be taken to make sure component works correctly if the time to process all auctions appears to be larger than the scheduled period (i.e. when the jobs can overlap in time). Since the job instances will run in multiple threads they need to synchronize to not process the same auctions at the same time.

1.2.7 Performance

Performance is an important requirement for this component.

DB operations have the most dramatic effect on performance so the number of DB requests need to be minimized and the queries need to be written efficiently. Here are some considerations that designers need to implement:

1. In section 1.2.4.1 a call to the ResourceManager#getAllResourceRoles() method is performed and then the resource roles are stored in a map. The resource roles are considered static (meaning they will never change during the application lifetime) so there's no need to call getAllResourceRoles() method more than once. The map should be created and cached only once.
2. In section 1.2.4.1 a call to the PhaseManager#getPhases() method is performed for every review review application and every resource role within that application. This call is also performed in section 1.2.4.2 for every review application but this method also changes the project phases. This is rather inefficient, so the call to the PhaseManager#getPhases() method should be performed only once per auction before processing the approved applications (but don't do the call if there were no approved applications). The phase extensions will change the phases timeline but that's ok since it doesn't change the phase IDs, types and statuses.
3. In sections 1.2.4.1 and 1.2.5 a call to the ProjectManager#getProject() method is performed but there's no need to make multiple calls for the same project. The call should be done only once per auction.
4. Manager instances like ProjectManager, PhaseManager, ResourceManager should be created and cached once. Don't create them on each job iteration.

1.2.8 Logging

The component will log its actions and errors at an appropriate level of details.

The component uses Logging Wrapper v2.0.0 to perform the logging so the log factory implementation needs to be set explicitly in the code when the application starts. The Log4jLogFactory will be used for this component, however the log factory implementation needs to be configurable.

1.3 Required Algorithms

Designers have to describe the following algorithms:

1. Algorithm to retrieve the list of review auctions to be processed.
2. Algorithm to match applications to the review application roles.
3. Algorithm to add a member to an Online Review project.
4. Algorithm to extend phases.
5. Algorithm for preparing and sending the notification emails.

1.4 Example of the Software Usage

The component will be used to regularly monitor all open review auctions and assign members to the review positions.

1.5 Future Component Direction

Any enhancement must be approved either in forum or in email with managers to eliminate over-complicating the component with useless functionality. All performance optimizations are encouraged and do not require an explicit approval.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None, only API interface and command line interface will be provided.

2.1.2 External Interfaces

None.

2.1.3 Environment Requirements

- Development language: Java 1.5
- Compile target: Java 1.5, Java 1.6

2.1.4 Package Structure

com.topcoder.management.review.assignment

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- Time interval the jobs are scheduled at.
- Background flag.
- "Guard file" filepath.
- Review Assignment strategy implementation.
- The phase type IDs and hours for the phase extension rules.
- Email templates, email subjects and sender's email address for each notification type. The templates should be read form text files.
- List of resource role IDs to send the "PM notification email" too.
- Log factory implementation.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None.

3.2.2 TopCoder Software Component Dependencies:

- Base Exception 2.0
- Configuration API 1.1.0
- Configuration Persistence 1.0.2
- Email Engine 3.2.0
- Document Generator 3.1.1
- Project Phases 2.0.2
- Project Management 1.2.2
- Project Management Persistence 1.2.3

- Phase Management 1.1.0
- Phase Management Persistence 1.1.0
- Job Scheduling 3.2.0
- Search Builder 1.4.1
- Command Line Utility 1.0.0
- DB Connection Factory 1.1.0
- Object Factory 2.2.0
- Resource Management 1.3.0
- Resource Management Persistence 1.2.4
- Review Application Management 1.0.0
- User Project Data Store 1.1.2
- Logging Wrapper 2.0.0
- Forums EJB (not a component)

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

Any third party library needs to be approved.

3.2.4 QA Environment:

- Java 1.5
- RedHat Linux 4
- 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.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification
- Notification email templates

3.4.2 Help / User Documentation

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