LEADS 2.0 EJB Report, Configuration, and Quartz Services Requirements Specification

1. Scope

1.1 Overview

The Leadership Acceleration and Development System 2 (LEADS 2) applications provides managers with the means to assess and predict their employee's leadership potential. It provides the capability to assign employees to specific leadership pipelines and to predict the employee's ultimate potential.

The purpose of the architecture is to define the business service layer for the application to perform the following tasks

- Allow a manager to assess the progress of employees by answering questions and then calculating their progress
- Manage various entities, such as profiles, questions, pipelines and eligible populations

1.1.1 Version

1.0

1.2 Logic Requirements

The purpose of this component is to provide services for managing configuration values in the system, getting reports. This component will also provide the Quartz Job that will be responsible for automating process pipeline identification cycling. This will also require a daemon class to allow the jobs to be run.

This component will provide the interfaces defined in the Report and Configuration Services Class Diagram. It will also provide local stateless session bean implementations for the interfaces. An ERD is provided.

This component will also provide the classes defined in the Quartz Daemon and Job Interface Diagram. These classes will be POJOs, not EJBs.

All operations in the implementations will operate against the DB2 database using JPA. All primary IDs are auto-generated.

1.2.1 ReportService

This service will retrieve the requested reports. The ARS contains the necessary details for which fields are retrieved and the search conditions. The following sections will state which ARS sections are applicable and which DB fields are to be used.

1.2.1.1 Searching for all employees of a manager in an organization

When searching for the employees in the organization, the search must be recursive. A manager may have several employees, who in turn may be managers of other employees, etc. We need all of these employees. The following algorithm should be used

1. do search for all employees under a specific manager (by his cnum) with LDAPUserService.searchUsers
2. Add the employees to a running list
3. For each employee

3.1. Do search for all employees under this employee as if he was a manager (by his cnum) with LDAPUserService.searchUsers

3.2. Use EmployeeProfileService.getEmployeeProfilesByManager to see if the employee is a matrix manager to any.

3.3. Add all retrieved employees to the running list

3.4. Repeat 2 for each retrieved employee in the last run

By the end of this process we will have one list of all employees that are directly or indirectly under the given manager.

1.2.1.2 getDirectNineBoxReport

This method will generate a direct 9 Box Report, as per ARS 2.5.

- Get all employees with EmployeeProfileService.getEmployeeProfilesByManager using this manager's cnum and ManagerRole.DIRECT. Ignore the sorting parameters.
- Get this employee's details with LDAPUserService using the employee's cnum.
- Combine each employee result into an EmployeeDTO, and return these DTOs as a list.

1.2.1.3 getOrganizationNineBoxReport

This method will generate an organizational 9 Box Report, as per ARS 2.5.

- Get this manager's details with LDAPUserService using the passed managerCNUM
- Search for all employees in the system using the LDAPUserService.searchUsers with UserSearchFilter for the given organization and employee roles. This search is recursive as per 1.2.1.1.
- For each employee
    - Get employee with EmployeeProfileService.getEmployeeProfile(String)
    - If employee is not of the desired band, ignore it
    - Using LDAPUserService, get the organization of the matrix manager. If it is of a different organization than the calling manager, ignore it
    - If filterManagerCNUM is provided, then if the employee does not have this manager in either the BP or Matrix roles, ignore it
    - Using LDAPUserService, get the employee's details.

    - Combine each employee result into an EmployeeDTO
- Return the EmployeeDTOs as a list

1.2.1.4 getSummaryReport

This method will generate a Summary Report, as per ARS 2.8.

If the report is direct:

- Get all employees with EmployeeProfileService.getEmployeeProfilesByManager using this manager's cnum and ManagerRole.DIRECT. Use the sorting parameters.
- Get this employee's details with LDAPUserService using the employee's cnum.
- Combine each employee result into an EmployeeDTO, and return these DTOs as a list.

If the report is organizational:

- Get this manager's details with LDAPUserService using the passed managerCNUM
- Search for all employees in the system using the LDAPUserService.searchUsers with UserSearchFilter for the given organization and employee roles. This search is recursive as per 1.2.1.1.
- For each employee
    - Get employee with EmployeeProfileService.getEmployeeProfile(String)
    - Using LDAPUserService, get the organization of the matrix manager. If it is of a different organization than the calling manager, ignore it
    - Using LDAPUserService, get the employee's details.
    - Combine each employee result into an EmployeeDTO
- Return the EmployeeDTOs as a list

1.2.1.5 getDirectRollupReport

This method will generate a direct Roll-up report, as per ARS 2.11

- Get all employees with EmployeeProfileService.getEmployeeProfilesByManager using this manager's cnum and ManagerRole.DIRECT. Use the sorting parameters to sort by bandId
- These employees will be grouped by their bandID. Each band will have a separate ReportRecord
- For each band ID group of employees
    - Set band to ReportRecord.Band
    - Count total number of employees in group ReportRecord.totalEmployees
    - Count total amount that have submitted (employee.status is Assessed) ReportRecord.submittedEmployees
    - Count total amount that have not submitted (employee.status is Not Assessed) ReportRecord.notSubmittedEmployees
    - Count total amount that are BTLR (employee.bltResourceIdentifier = true) and have submitted ReportRecord.btlrEmployees
    - Count total amount that are not BTLR (employee.bltResourceIdentifier = false) that

have submitted ReportRecord.notBtlrEmployees
    - Set ratio of (btlrEmployees / submittedEmployees) to ReportRecord.btlrPercentage
- Set grand employee total to Report.totalEmployees
- Set grand submission total to Report.submissions
- Set Report.completePercentage to Report.submissions/ Report.submissions
- Set all above records to Report.records
- Sort report as per user request

1.2.1.6 getOrganizationRollupReport

This method will generate an organizational Roll-up report, as per ARS 2.11

- Get this manager's details with LDAPUserService using the passed managerCNUM
- Search for all employees in the system using the LDAPUserService.searchUsers with UserSearchFilter for the given organization and employee roles. This search is recursive as per 1.2.1.1.
- For each employee
    - Get employee with EmployeeProfileService.getEmployeeProfile(String)
    - Using LDAPUserService, get the organization of the matrix manager. If it is of a different organization than the calling manager, ignore it
    - If filterManagerCNUM is provided, then if the employee does not have this manager in either the BP or Matrix roles, ignore it
- The resultant list of employees is assembled into a report the same way as shown in 1.2.1.4 above.
- Sort report as per user request

1.2.1.7 getHighLevelSummaryReport

This method will generate a High-level Summary Report, as per ARS 2.14.

- Get the employees using the search filter with EmployeeProfileService.searchEmployeeProfiles
- The resultant list of employees is assembled into a report the same way as shown in 1.2.1.4 above.
- Sort report as per user request

1.2.2 SystemConfigurationPropertyService

This service will work with the SystemConfigurationProperty entity. It will operate against the system_configuration table.

The getPipelineCycleStatus and updatePipelineCycleStatus methods are simply convenience methods for managing the value of a specific property. The actual name of this property may be configurable.

1.2.3 *TogglePipelineIdentificationCycleJob*

This job will be simply told to what state to update the pipeline cycling. It will use the SystemConfigurationPropertyService for this.

1.2.4 *TogglePipelineIdentificationCycleDaemon*

The toggling job is expected to be run under a scheduler, which should automatically run on scheduled time. Use Quartz scheduler here (as it has proved to be quite robust for job scheduling).

Job schedule (i.e. when the job will fire) shall be configurable for this job. The Quartz framework. Scheduler (the daemon) will be designed and should be modifiable.

1.2.5 *Model*

This component will use the model defined in the Model and Exceptions component and can be seen fully in the Model and Exception Class Diagram.

1.2.6 *Exceptions*

Each service will use the LeadsServiceException as its top-level application exception. This exception is annotated with @ApplicationException, so if an exception occurs, the transaction will be rolled back automatically.

Similarly, if any service interface implementation requires any initialization beyond that which is done by simple injection can use the LeadsServiceConfigurationException for that purpose.

If a service requires additional exceptions, it will extend the above exceptions as applicable.

1.2.7 *Persistence*

This component will use JPA (with Apache OpenJPA 1.2.1: http://openjpa.apache.org/) to interact with a database.

This component will provide the necessary mapping file for accessing the DB via JPA.

1.2.8 *Logging*

The services will log activity and exceptions using the Log4j in this component.

It will log errors at Error level, and method entry/exit information at DEBUG level.

Specifically, logging will be performed as follows, if logging is turned on.

- 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.
- Method request and response parameters will be logged with DEBUG level
  - Format for request parameters: [Input parameters[{request_parameter_name_1}: { request_parameter_value_1}, {request_parameter_name_2}: { request_parameter_value_2}, etc.)]]
  - Format for the response: [Output parameter {response_value}] . Only do this if there are no exceptions and the return value is not void.
  - If a request or response parameter is complex, use its toString() method. If that is not implemented, then print its value using the same kind of name:value notation as above. If a child parameter is also complex, repeat this process recursively.
- All exceptions will be logged at ERROR level, and automatically log inner exceptions as well.
  - Format: Simply log the text of exception: [Error in method {*className.methodName}*: Details {*error details*}]

In general, the order of the logging in a method should be as follows:

1. Method entry
2. Log method entry
3. Log method input parameters
4. If error occurs, log it and skip to step7
5. Log method exit
6. If not void, log method output value
7. Method exit

1.2.9 *Configuration*

Configuration will be done via injection.

1.2.10 *Transactions*

The services will use container managed transaction. And the methods that will change the database data will be annotated to indicate that the transaction is required.

```
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
```

1.2.11 *Thread-safety*

The services will be effectively thread-safe. It can be assumed that the entities being persisted won't change during a persistence operation (for example, only one thread will work with a given Campaign entity instance at a time), but with that exception, the rest of the code will be able to be

called from multiple threads.

1.2.12 *Auditing*

None

1.2.13 *Method argument handling*

This section describes the generic convention for expected handling of arguments, and may not refer to any specific operation for this component. However, any operation defined in this component that is affected by these conventions may override part or all of this. Such an override will be explicitly mentioned in the operation descriptions above.

- If getting an entity and it is not found, a method will return null.
- If getting a list of entities, and none are found, a method will return an empty list.
- Any list handled by the component must not contain null elements.
- Unless stated, all input arguments will be required (non-null). String arguments must not be null/empty.

If any input parameter requirement is not met, an IllegalArgumentException will be thrown.

1.2.14 *Avoiding name collisions*

Please be aware that there will be other service classes from other components in the services package, so please use unique and specific names for any base or helper classes so that they will not conflict with other such classes in other components for this architecture.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

This component will be used to manage system configuration values, get reports. This component will also be used to schedule a job for updating the pipeline cycling status.

1.5 Enhancement policy

In order to eliminate superfluous, useless, and/or bloated enhancements from the application, the following policy on enhancements is in effect for this competition.

All major enhancements must be explicitly approved by the architect (the approval of PM and/or co-pilot is not sufficient). All enhancements proposed in the future direction section are considered to be approved. Only if the architect approves the enhancement may it be added to a design. Any attempt to add a major enhancement to a design without this approval will result in

that enhancement to not be eligible for a score of 4 in the requirements section (unless this idea happens to correspond to another submission's enhancement that was approved).

You may outline the enhancement proposal in the forum. You may also contact the architect directly to retain the privacy of your ideas. After the conclusion of the submission phase, the architect will notify the reviewers of the approval so they may score for it.

Be aware that the approval of an architect does not automatically assure a 4 in the requirements section. The architect will approve an enhancement or enhancements based on how useful and pertinent they are to the application. The reviewers, though, will decide if the enhancement or sum of enhancements is substantial. It is possible that the architect may advise the reviewers of how substantial they may be to the application, but the final decision will be in the hands of the reviewers.

When making an enhancement request via Contact Manager, please put the following in your first line:

Enhancement Request

At this time, it may also help to contact this architect directly with Member Contact since Contact Manager does not send a notification to the architect. This would most likely expedite the process.

1.6 Future Component Direction

None

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

The design must adhere to outline of the Report and Configuration Services Class Diagram and the Quartz Daemon and Job Interface Diagram.

2.1.3 Environment Requirements

- Development language: Java 1.6, J2EE 1.5
- Compile target: Java 1.6, J2EE 1.5
- WebSphere Application Server ND 7.0
- DB2 for z/OS version 9, New Function Mode

2.1.4 *Package Structure*

hr.leads.services

3. Software Requirements

3.1 Administration Requirements

3.1.1 *What elements of the application need to be configurable?*

Each service will be configured with the following

- EntityManager
- Logger

In addition, the SystemConfigurationPropertyService will have

- PipelineCycleStatus configuration parameter name

3.2 Technical Constraints

3.2.1 *Are there particular frameworks or standards that are required?*

- J2EE 1.5
  - EJB 3.0
  - JPA 1.0
- Spring 2.5
- IoC

3.2.2 *Component Dependencies:*

- LEADS 2 EJB Model and Exceptions 1.0

**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:*

- JPA 1.0 (with Apache OpenJPA 1.2.1: http://openjpa.apache.org/)
- Log4j 1.2.15

3.2.4 *QA Environment:*

- Java 1.6/J2EE 1.5
- WebSphere Application Server ND 7.0
- DB2 for z/OS version 9, New Function Mode
- LDAP
- JPA 1.0 (with Apache OpenJPA 1.2.1: http://openjpa.apache.org/)
- Spring 2.5.6
- Quartz 1.8.3

- Log4j 1.2.15

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines.

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 the TopCoder UML Tool.