IM Application Logic 1.0 Requirements Specification

1. Scope

1.1 Overview

The IM Application Logic component performs the backend logic for a client-server chat application. The logic is event-driven. It provides implementations for all the necessary interfaces in order to handle the events properly. These implementations will be plugged into the appropriate places during the initialization of the application.

1.2 Logic Requirements

1.2.1 Change in User Status

The component will provide an implementation of the `ChatStatusEventListener` interface. Below is the logic when the user status changes:

- If it changes from OFFLINE to ONLINE, register a user message pool for the user.
- If it changes to OFFLINE, unregister the user message pool for the user.

1.2.2 Change in Session Status

The component will provide an implementation of the `ChatStatusEventListener` interface. Below is the logic when the user status changes:

- If it changes to OPEN, and it is a 1-1 session or Sales session (which means two users were added and requires acknowledgement), post the Enter Chat Message directly to both users. For all session types, do the same thing as when a user is added for each user of the session (see 1.2.3).

1.2.3 Change of User in Session

The component will provide an implementation of the `ChatSessionEventListener` interface. Below is the logic when the user status changes:

- When a user is requested, post the Ask For Chat Message directly to the user.
- When a user is added to session, and if the session status is OPEN, register a session message pool for the user. Post Presence Message of other users to this user, and post Presence Message of this user to other users.
- When a user is removed from session, and if the session status is OPEN, unregister the session message pool for the user. Post Presence Message of this user to other users. For any session status, if the removed user is the last user of the session, change the session status to CLOSE.

1.2.4 Service Logic

The component will provide an implementation of the `ServiceHandler` interface. The requester object should wrap the user id and session id. The responder object should wrap the user id. Both wrapper classes should be defined in this component.

Logic for *to-be-serviced*:

- Request the responder to the session.
- If the responder does not accept the request in a configurable period of time, request the remaining responders of the same category to the session.
- If still no responder accepts the request in that period of time, move it to the servicing state with no responder.

Logic for *servicing*:

- If there is a responder, add the responder to session, and update the session status to OPEN. Post Session Unavailable Message to other responders requested before. Depending on the implementation, this should also notify the handler that some responder accepts the request.
- If there is no responder, post Session Unavailable Message to the requester to notify him service is not available. Remove the requester from session. Change the user status to OFFLINE.

1.2.5 Inactivity Logic

The component should detect inactivity of the message pools. It searches for the message pools that have not been pulled in a configurable time period. Below is the logic when the message pool is inactive:

If session message pool is inactive,

- Remove user from session

If user message pool is inactive,

- Remove the user as requester and responder from the service engine
- Remove user from all sessions
- Change user status to OFFLINE

A command line interface is required. The Job Scheduling will be used to run the job for inactivity detection at configurable intervals.

1.2.6 Logging

Logging should be performed for each operation using the Logging Wrapper. Logging must comprise user ID, timestamp, action taken, and entity IDs effected.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The IM application will use this component to perform the backend logic. The listeners and handlers will be plugged into the corresponding places to make the application work as a whole.

1.5 Future Component Direction

None

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

None

2.1.3 Environment Requirements

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

2.1.4 Package Structure

- com.cronos.im.logic

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- Time allowed for responder to accept the chat
- Inactive time period
- Time interval to run the scheduled job

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None

3.2.2 TopCoder Software Component Dependencies:

- Job Scheduling 2.0
- Service Engine 1.0
- Chat User Profile 2.0
- Chat Session Manager 1.0
- Chat Status Tracker 1.0
- Chat Message Pool 1.1
- IM Messenger 1.0
- Logging Wrapper 2.0

**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:

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.

3.4 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.