System Design Exercise: Chat System (Solution)

Contents

1 Subsystem Decomposition

1.1 Overview

Figure 1 shows the subsystem decomposition of the chat system ICU.

The architecture mainly follows the client/server style. In particular, account administration (e.g., registering accounts, login), history administration (e.g., saving and searching histories) and the establishment of connections between two chat clients are services that always go through the server.

Once a connection has been established between two chat clients, the communication follows the peer-to-peer style if both clients use the standalone application. For security reasons and limitations of web applications, if one or both of the chat clients use the web interface, the communication still goes through the server.

Server. The server has been split into two main components: one handling account information, the other handling history information. Thereby, users' personal information and chat messages are stored separately. The purpose of this separation is to facilitate the privacy policy of the chat system.

The design includes a layer to decouple the system from the actual database deployed, thereby allowing simpler switching to other databases.

Client. The client consists of a component that contains the business logic and two interfaces to the logic. One interface allows one to use the chat system as a standalone application, while the other allows one to use it from a web browser.

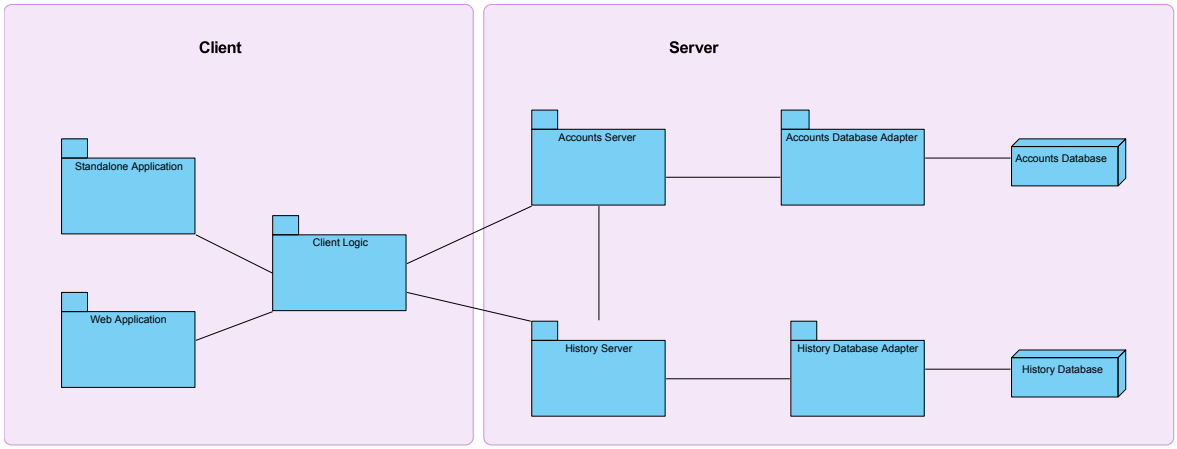

Figure 1: UML Package diagram of subsystem decomposition

1.2 Subsystem Interfaces

The following describes the way subsystems communicate with each other.

Client Logic and **Accounts Server**, and
Client Logic and **History Server**: Java RMI.

Accounts Server and **Accounts Database Adapter**, and
History Server and **History Database Adapter**: Local method invocations.

Accounts Server and **Accounts Database Adapter**, and
History Server and **History Database Adapter**: SQL queries.

Accounts Server and **History Server**: Java RMI.

Client Logic and **Client Logic**: once a connection between two standalone chat applications has been established, the connection is peer-to-peer. The connection is maintained by Java RMI.

Standalone Application and **Client Logic**: The application provides a GUI to view information received from the logic (via local method calls) and to convey user commands to the logic (via events).

Web Application and **Client Logic**: The application is a Java applet embedded in an HTML page. The applet provides a GUI for the limited services of the web application listed

below. As for the **Standalone Application**, communication happens via local methods and events.

2 Description of Subsystems

2.1 Accounts Server

Subsystem purpose. Responsible for the management of account information and synchronization between account information and history database.

Provided services.

- *Add account*: save new account information upon user registration
- *Remove account*: remove existing account information and related chat histories upon user deregistration
- *Login*: check if a given username/password pair is valid, and if so, register user as being online
- *Logout*: register given user as being offline
- *Modify personal information*: modify information about existing user
- *Modify friends list*: add usernames to or remove usernames from a user's friends list and send a notification to all online users who are affected by the modification
- *Search in account data*: search among account data based on a given keyword

Used services.

- **History Server**::*Remove history*: the service is used upon deregistration of a user in order to remove all messages in stored chat histories that are related to the user being deregistered (service *Remove account*)
- **Client Logic**::*Receive notifications about friends-list changes*: used to inform users whenever their friends lists have been updated (service *Modify friends list*)
- **Client Logic**::*Receive notifications about online-friends changes*: used to inform users whenever a friend logs in or out (services *Login*, *Logout*, *Remove account*, and *Modify friends list*)
- The services of the **Accounts Database Adapter**: whenever data in the **Accounts Database** is to be modified or queried.

Threads. Each RMI call to the server spawns a separate thread.

2.2 History Server

Subsystem purpose. Responsible for conveying chat messages to their specified receivers and for the management of chat histories.

Provided services.

- *Convey message*: send a message received from some user to another user, and enter the message into the database
- *Remove history*: remove all messages in which a given user participated
- *Add history*: add given bulk of chat messages to the database (used when histories stored locally on client machines are sent to the server)
- *Get history*: provide chat messages of a given user after a given point of time

Used services.

- **Client Logic**::*Get message*: used when a chat message is to be sent to a given user (service *Convey message*)
- **Client Logic**::*Upload history*: used when locally stored histories are to be sent from the client to the server (service *Add history*)
- The services of the **History Database Adapter**: whenever data in the **History Database** is to be modified or queried.

Threads. Each RMI call to the server spawns a separate thread.

2.3 Accounts Database Adapter

Subsystem purpose. An adapter to translate the high-level tasks of the Accounts Server that involve the database into vendor specific SQL. By replacing the adapter one may change the underlying DBMS.

Provided services. Modifying and querying the **Accounts Database**.

Threads. Each interaction with the database is performed in a separate thread.

2.4 History Database Adapter

Subsystem purpose. An adapter to translate the high-level tasks of the **History Server** that involve the database into vendor specific SQL. By replacing the adapter one may change the underlying DBMS.

Provided services. Modifying and querying the **History Database**.

Threads. Each interaction with the database is performed in a separate thread.

2.5 Accounts Database

The **Account Database** is an external component for storing account informations.

2.6 History Database

The **History Database** is an external component for storing chat histories for users that have an account in the system.

2.7 Client Logic

Subsystem purpose. Provide main services for accounts management and chat functionalities on the client.

Provided services.

- *Register*: register a new user of the system
- *Deregister*: deregister an existing user of the system
- *Login*: log an existing user in to the system
- *Logout*: log an existing user out of the system
- *Send message*: send a chat messages to another user of the system
- *Get message*: receive a message from another user of the system
- *Upload history*: send a bulk of chat messages (histories) that have been stored locally so far (during chats over peer-to-peer connection)
- *Modify personal information*: modify previously given personal information of an existing user
- *Modify friends list*: add or remove users from the list of friends for a given user
- *Receive notification about friends-list changes*: receive notification if new friendship has been initiated by an other user or an existing friendship has been deleted
- *Receive notification about online-friends changes*: receive notification whenever a friend logged in or logged out
- *Search in accounts data*: search in the personal information of users based on a keyword (e.g., to find friends)
- *Search in chat histories*: search in stored (local and on server) chat histories in which given user participated

Used services.

- **Accounts Server**::*Add account*: used upon registration of a new user (service *Register*)
- **Accounts Server**::*Remove account*: used upon deregistration of an existing user (service *Deregister*)
- **Accounts Server**::*Login*: used upon user login (service *Login*)

- **Accounts Server**::*Logout*: used upon user logout (service *Logout*)
- **Accounts Server**::*Modify personal information*: used when a user modifies his/her personal information (service *Modify personal information*)
- **Accounts Server**::*Modify friends list*: used when a user modifies his/her friends list (service *Modify friends list*)
- **Accounts Server**::*Search in account data*: used when a user initiates a search among accounts data (service *Search in accounts data*)
- **History Server**::*Convey message*: used when a user sends a chat message to another user and the connection is client/server style (service *Send message*)
- **History Server**::*Add history*: used when locally stored chat histories are sent to the **History Server** (service *Load history*)
- **History Server**::*Get history*: provide chat messages of a given user between a given period of time (service *Search in chat histories*)
- **Client Logic**::*Get message*: used when user sends a chat message to another user and the connection is peer-to-peer between the users (service *Send message*)

Threads. Each in-coming RMI call spawns a separate thread. To prevent the GUI to be blocked upon time-consuming operations, each GUI event spawns a separate thread.

2.8 Standalone Application

Subsystem purpose. Provides the interface for the chat system over a standalone Java application.

Provided services. Provides a GUI for the services of the **Client Logic**.

Used services. Uses all services of the **Client Logic** whenever any of these services is initiated by the user through the GUI.

Threads. Upon start-up of the application, a thread is spawn for the main GUI frame of the application.

2.9 Web Application

Subsystem purpose. Provides the interface for the chat system as a web application. The application is a Java applet that is embedded in an HTML page.

Provided services. Provides a GUI for limited services of the **Client Logic**, namely: *Login*, *Logout*, *Send message*, and *Get message*.

Used services. Uses the services *Login*, *Logout*, *Send message*, and *Get message* of the **Client Logic** whenever any of these services is initiated by the user through the GUI.

Threads. Upon start-up of the application, a thread is spawn for the main GUI frame of the application.