Design Milestone 2

Alvaro Morales, Ayesha Bose, John Holliman

{alvarom, aybose, holliman}@mit.edu

Revised Server Design

`class Server`

Starts a ServerSocket on the default server port. Listens for connections, and starts a User thread for every client that connects. It keeps a blocking queue instance, and passes it to every User.

`class User implements Runnable`

The class first authenticates the connected client, by listening for a Login request and processing it. The first request must be a Login request.

Then, the User listens for any more requests. Upon receiving a request, it adds it to the blocking queue.

`class ChatRoom`

Represents a chat room. Keeps a list of Users connected to the chat room. Has methods to push a Response to connected users.

`class RequestHandler implements Runnable`

Class in charge of processing requests and emitting responses. Keeps a list of active chat rooms, and a blocking queue for processing requests. Takes a request from the blocking queue, and atomically processes it. Broadcasts a Response to all users in the chat room. Listens for more Requests.

Concurrency Strategy

Server Side

We will use a LinkedBlockingQueue implementation for the request blocking queue. This implementation is thread-safe, and allows multiple Users to put Requests onto it without running into race conditions, messing up the order of requests or overwriting requests. These methods will be atomic.

The request handler class will contain an overloaded handleRequest method (one for each type

of requests, e.g. Login, SimpleMessage,etc). Each of these methods will be synchronized. The running RequestHandler thread will take from the queue, and run the corresponding handleRequest method.

Because we're using a data structure that is thread-safe, and building a thread-safe RequestHandler class that contains synchronized, atomic methods, our server design is thread-safe.

Client Side

The client concurrency strategy is similar to that of the server. When the client connects to the server two things are created: a ChatSession, responsible for keeping track of the ChatWindows the client has open and maintaining a response blocking queue, and a ResponseHandler, responsible for handling the Response objects.

As with the RequestHandler on the server side, the ResponseHandler will contain overloaded handleResponse methods. Each of these methods will be synchronized. The ResponseHandler thread takes Responses from the blocking queue in the ChatSession and runs the corresponding handleResponse method.

Most of the Response objects will require that we mutate the ArrayList of messages in a specific ChatWindow and write to the corresponding JTextArea in the GUI. This is all handled by synchronized methods in specific ChatWindow class. With the exception of System Messages and help message pre connection, all writing to the main chat window will be done by active ChatWindow or the ChatSession (in the case of adding new ChatRooms to the Chat Rooms list).

When the user elects to terminate a ChatWindow, we will join all of the threads in the ChatSession blocking queue to ensure that none of those threads can modify the specified ChatWindow after it is terminated. The same is done when the entire ChatSession is terminated.

Testing Strategy

- Connect to Chat Server
- Create a username
- Create a new room
- Type message into room
- Leave room

This test should show the message displayed next to the specified username, the new chat room should be in the list of available chat rooms. The user should then be able to leave the room.

- Connect to Chat Server
- Create a username
- Create a new room
- Type message into room
- Leaves room
- Other user connects to chat server
- Other user creates a username
- Other user creates a new room
- Other user types message into new room

This test should show the message displayed next to the first specified username, the new chat room should be in the list of available chat rooms, the other user should be able to create a new username and login to the chat server and not see the other user's chat room in the list, the new chat room that the second user creates should join the list of available chat rooms, the message the second user types should be only displayed in the second user's chat room next to the second user's name.

- Connect to Chat Server
- Create a username
- Create a new room
- Type message into room
- Other user connects to chat server
- Other user creates a username
- Other user joins existing room
- Other user types message into new room
- First user types message into room
- First user leaves room

- Second user leaves room

This test should show the message displayed next to the first specified username, the new chat room should be in the list of available chat rooms, the other user should be able to create a new username and login to the chat server and see the other user's chat room in the list. The second user should be able to join the first user's room. The first user should be notified that the second user joined. The second user should not be able to see the first user's previous message. The second user should be able to type in a message, which will be displayed next to their username. The first user should see this new message. The first user should be able to type in a message, which will be displayed next to their username. The second user should see this new message. The second user should be notified that the first user left. The second user should be able to leave.

- Connect to Chat Server
- Create a username
- Create a new room
- Type message into room
- Other user connects to chat server
- Other user creates a username
- Other user joins existing room
- Other user types delayed message into new room
- First user types message into room
- First user leaves room
- Second user leaves room

This test should show the message displayed next to the first specified username, the new chat room should be in the list of available chat rooms, the other user should be able to create a new username and login to the chat server and see the other user's chat room in the list. The second user should be able to join the first user's room. The first user should be notified that the second user joined. The second user should not be able to see the first user's previous message. The second user should be able to type in a delayed message, which will be displayed next to their

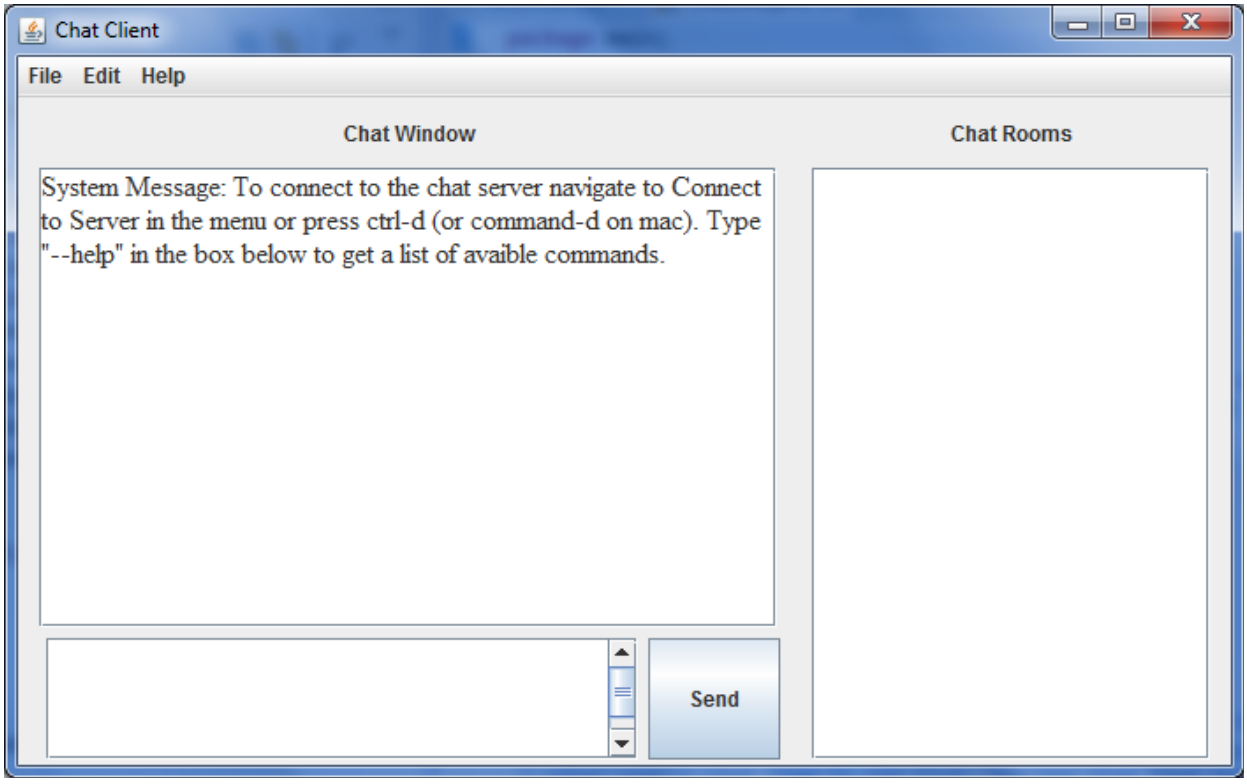
username immediately. The first user should be able to type in a message before the second user's message shows up for them, which will be displayed next to the first username. The second user's delayed message should show up before the first user's message. Both users should be able to see both messages eventually.The second user should be notified that the first user left. The second user should be able to leave.

Other tests:

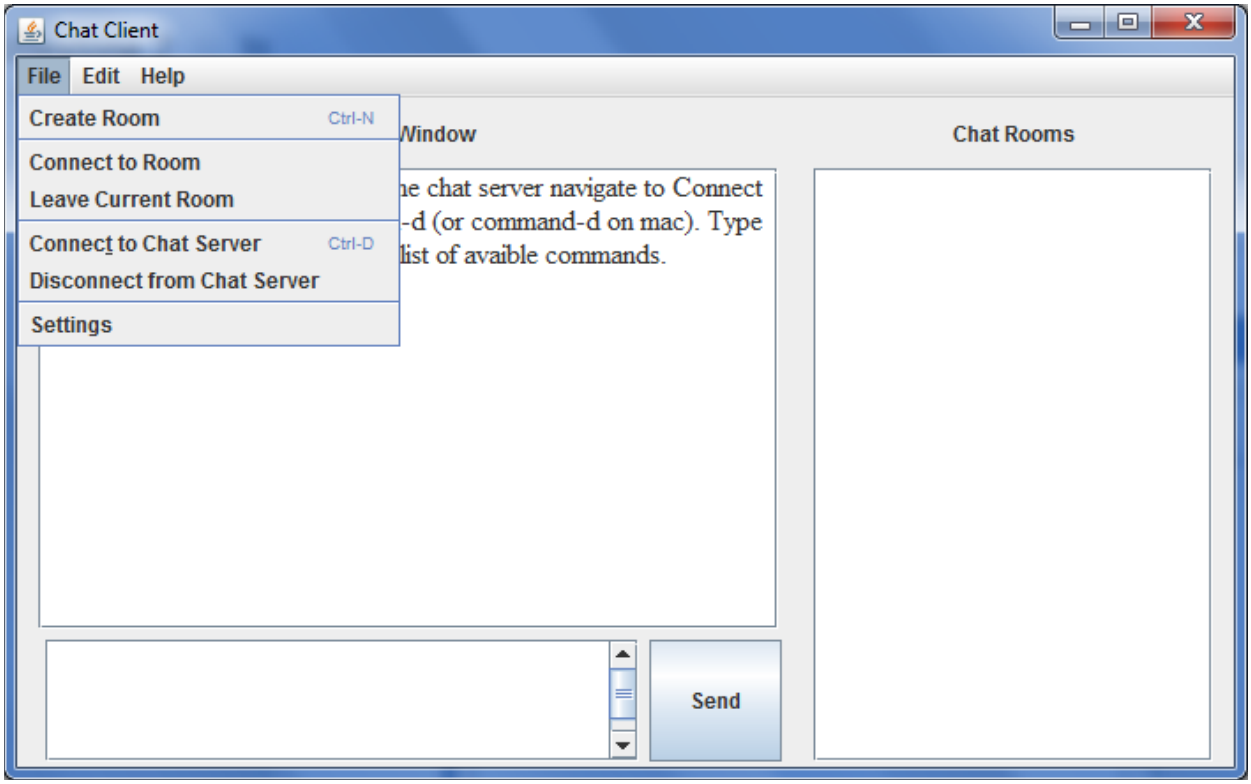
- One user, multiple chat rooms
- Multiple users, one chat room
- Multiple users, multiple chat rooms

GUI Screenshots

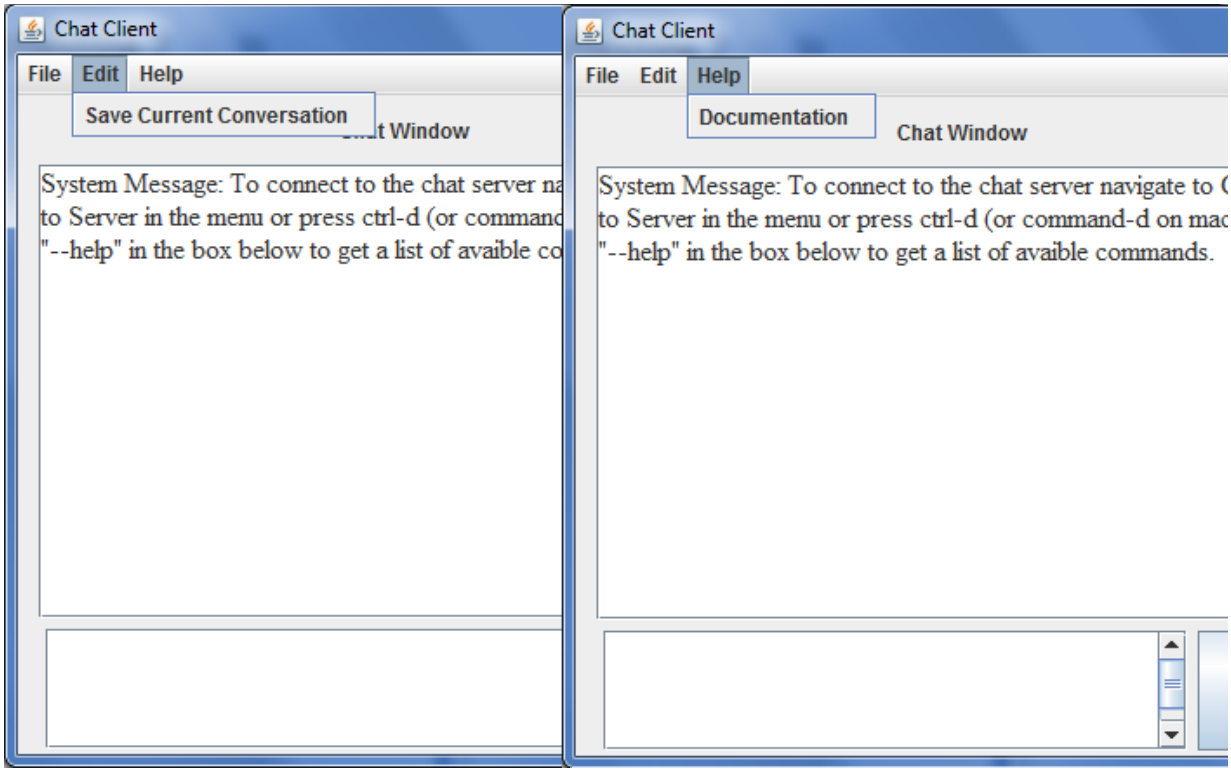
Main page:



File menu:

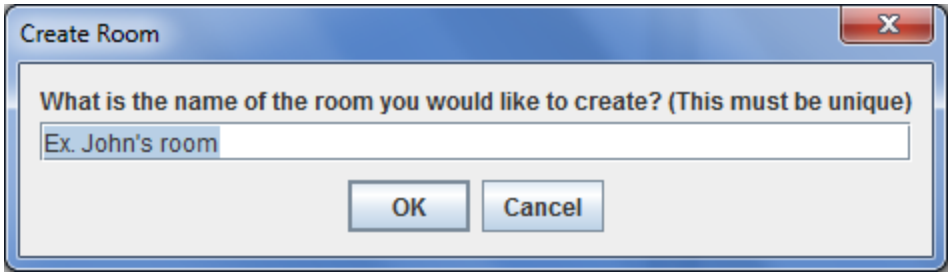

Chat Client
File
Edit
Help
Create Room
Ctrl-N
Connect to Room
Leave Current Room
Connect to Chat Server
Ctrl-D
Disconnect from Chat Server
Settings
Window
Chat Rooms
he chat server navigate to Connect
-d (or command-d on mac). Type
list of avaible commands.
Send

Edit and Help Menus:

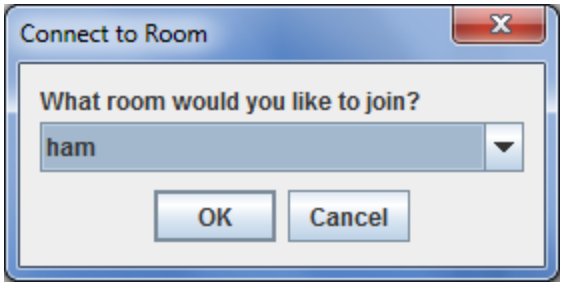


Connect to server:

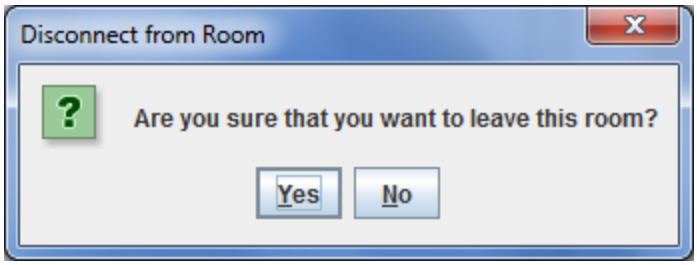
Chat room:



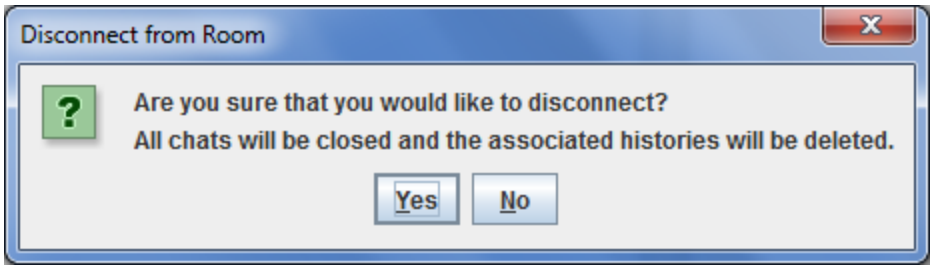
Connect to a room:



Leave a room:



Disconnect from the server:



Snapshot Diagram of Conversation (from milestone 1)