Design Document

Yuhuan Qiu (yq56)
Eric Wang (ew366)
Byungchan Lim (bl458)
Somrita Banerjee (sb892)

Logistics
Weekly regular meetings on
Tues: 9:00PM - 12:00AM
Wednesday: 8:30PM - 12:00AM
Friday: 5:00PM - 12:00AM
Saturday: 12:00PM - 12:00AM

1. System Description

Core vision

A customizable terminal based XMPP client/server system that provides basic messaging, and functionality expected of an IM application, as well as additional features such as small interactive games between users/groups.

Short bulleted list of key features

- Talk to an AI/Bot or a random person or direct message or join a chat room- When a user comes online, they have four options. They can choose to talk to a bot that responds to messages by picking from a stored list of responses.
- They can also choose to be paired up with a random person who is online or directly message another user. They can also join a chat room from a list of available chat rooms that are topic-based or interest- based to have a group chat.
- Support sending files- Users can send les to one another.
- Text formatting- Users can format their text, add background/foreground color, underline words, etc.
- Small Games - Support small interactive games between users within the application, i.e. Tic-Tac-Toe, Rock-Paper-Scissors
- Customizable Keybindings - To send last used message, Switch chats, See Favorites Default Vi-like or Emacs like configurations.

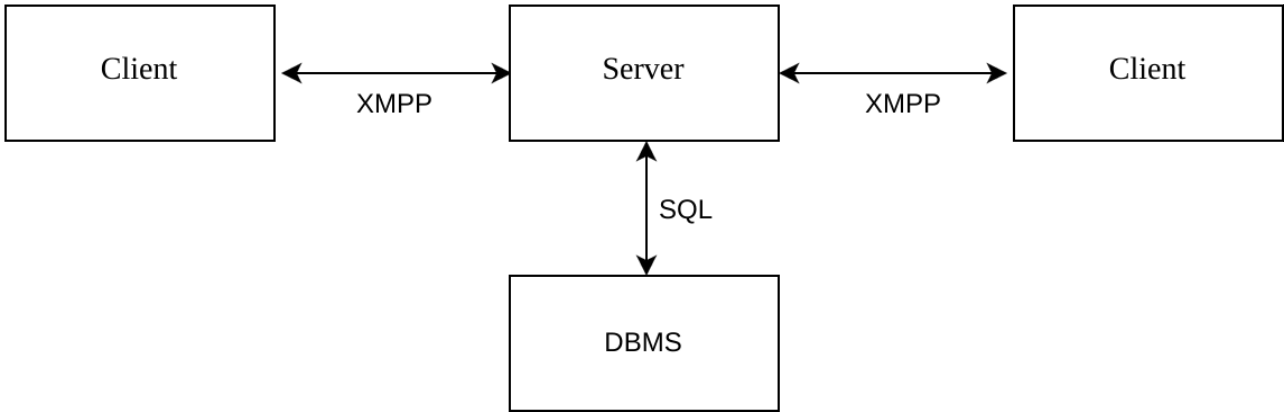
Narrative description

We intend to build a instant messaging client/server application implementing a XMPP inspired protocol using lwt, ocaml-sqlexpr, and ANSITerminal OCaml packages. There will be a central server to which users send requests to join a certain chat room, message a certain person, be paired with a random person or even talk to an AI/Bot. They will then be able to send formatted text messages to the room/person. Some other extended features are detailed in the list below. A

key innovative feature of our messenger application is that users have the ability to socialize with each other not only through text but also by playing small games with each other like Tic-Tac-Toe or Rock-Paper-Scissors. There will also be some customizable aspects to our messenger client, such as keybindings.

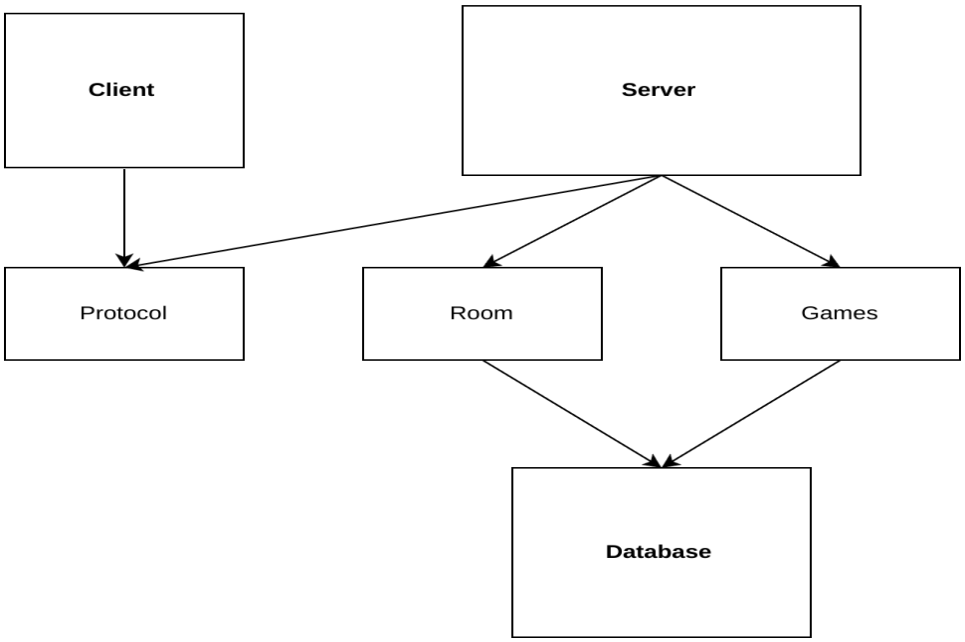
2. Architecture

Components and Connectors- Multi-tier client server architecture.
The components are the Server, a number of Clients, and a DBMS.
The communication between the servers and client is handled through XMPP and the communication between the server and the DBMS is through SQL.
C&C



3. System design

MDD



Modules:

Client: connects to a server, sends messages to server, receives messages from server
Steps (Simplified)
1. Create socket
2. Connects to server
3. Sends requests, receives responses

Server: communicate with clients, accepts client connection requests, processes requests, and sends messages to clients
Steps (Simplified)
1. Create socket
2. Set port to listen to
3. Accept connections
4. Receives requests/Sends responses

Protocol: Creates and parses messages or commands between the client and server. Includes the types of messages and commands (message ID, body, records/variants etc.) as well as the related functions required (e.g. to create a message body).

Room: Chatroom abstraction.
Includes room interface which has function to add/remove clients, broadcast message to everyone. We will have two functors- one will create a chatbot room with exactly one person and our chat bot AI (which is going to be a Drumpf bot). The other functor will create a normal chatroom with multiple people.

Database: Handles and processes requests concerning data stored in database. Inserts records or retrieves records.
Room and Game modules would call functions from database modules to call up relevant data when needed such as chat histories, favorite people (donald trump) list, game win/loss histories, etc.

Game: Handles game commands. Games are handled by chatbots, similar to Facebook Messenger's fbchess bot. Users present in a room with a game bot can enter in commands (e.g. @fbchess start, @fbchess Ne4), and then bot will post the updated game state in the chat. Games we are considering including are Tic-Tac-Toe and chess.

5. Data

Our system will maintain a SQLite database to hold the chat history for all users. One table uses the users as "keys" and stores their game history, i.e. average score and games played in the past, as well as game results. Another table also uses the users as "keys" and stores the list of people they've talked to in the last 24 hours. Similarly, we might have tables to store other attributes of the client. This database will be accessed by the server using the ocaml-sqlexpr library. This database is the primary persistent component state of the messenger.

One thing to note is that the database will be using the usernames of the clients as the key. This means that whenever a user logs on, regardless of what computer or IP address they use, they must use the same string as their username.

Format for communication: XMPP inspired. Instead of representing requests/responses as XML, we will represent it as OCaml records with XMPP inspired fields, i.e. message id, body, etc that normally make up a stanza. See the data structures within protocol.mli.

6. External Dependencies

We'll be using **Lwt** which is a cooperative multitasking library for handling asynchronous requests. We'll be using **ANSITerminal** for printing and for formatting the text. **OCaml-sqlexpr** for small database queries. We're also using a plugin to transfer files from one user to another, as well as a plugin to play chess in OCaml. **Ocamlnet's**
Ocaml-sqlexpr: https://github.com/mfp/ocaml-sqlexpr
Ocamlnet: http://projects.camlcity.org/projects/ocamlnet.html
File-transfer plugin: https://github.com/dannywillems/ocaml-cordova-plugin-file-transfer
Chess plugin: https://github.com/jaredwad/Chess

We might use **rlwrap** to allow the users to use the arrow keys to go up and down in the message thread.
We will also be including third party code that exists to play a game like chess that has a terminal mini-game built in. This is not a dependency but a (credited) inclusion of third-party source code with our own. We may be using the following as reference.
XMPP Book
http://oriolrius.cat/blog/wp-content/uploads/2009/10/Oreilly.XMPP.The.Definitive.Guide.May.2009.pdf

6. Testing Plan

Testing plan: *How will you test your system throughout development? What kinds of unit and module tests will you write? How will you, as a team, commit to following your testing plan and holding each other accountable for writing correct code?*

Testing plan: ***Agile testing***
The server module will be tested as soon as it is finished, and incrementally before it is finished, before the client is done, using telnet or nc (netcat) to connect remotely. Each function will be tested if possible as it's written, i.e. a test-driven development workflow. The same idea applies to the client module as well. For the other modules, unit testing the individual modules is straightforward.

As a team we will hold each other responsible of writing tested correct code by holding biweekly regular meetings and emphasizing to each other the importance of correct code that does what is specified by the interface. At the biweekly regular meetings, we will have code reviews. Additionally, we will finalize and merge git branches after code review.