Anthony Comstock

Term Project: *ChatApp*

Design Document

Table of Contents

1 Introduction

This document is intended to explain at first a high level and then in more detail the design of the ChatApp application. The first section explains the purpose, scope, target audience and terms and definitions. The next section explains the high level overview of the application and the subsystems of it. The last section goes into detail on the subsystems and lists their properties.

1.1 Purpose and Scope

This document lays out the design for the ChatApp application. It is designed to give both a overall high level design of the program as well as the low level design of each class used in the application.

1.2 Target Audience

This document it for myself, and my Teacher and TA in order to evaluate the design of this project.

1.3 Terms and Definitions

Throughout this document the word object refers to a instantiation of a class with data and operations. The term struct is used to mean a Java object that has only public fields and no operations on it. Users are the people who will be using this application by interacting with the client. The client is the program that will be on the users computer and connect to the server.

2 Design Considerations

Listed below are the constraints and dependencies that dictated many of the design choices as well as the methodology used throughout the design which further constrained how the project was put together and how the application was designed.

2.1 Constraints and Dependencies

The major constraint of this program is the language which must be Java. Other constraints include the time constraint which is the class term. Being that this program will be written wholly in Java it is dependent on the JDK and a Java interpreter.

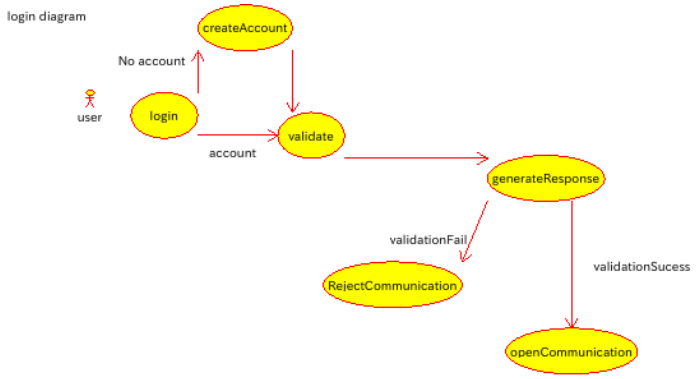
2.2 Methodology

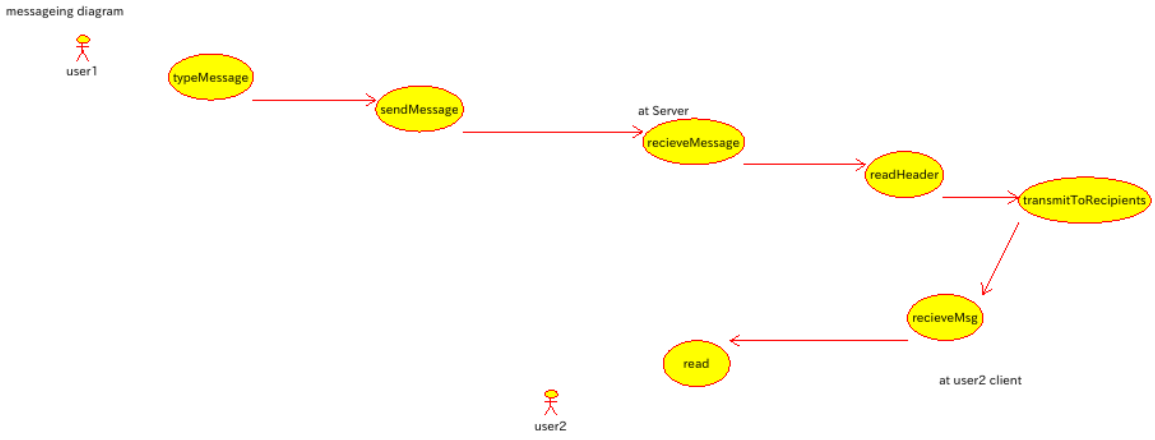
The software engineering methodology I am using for this project is based on the Rational Unified Process(RUP) Methodology. This is an iterative approach that allows for refinement over time as well as breaking the program down into small chunks and implementing them over many iterations. In the case of this project each chunk is a simple object.

3 System Overview

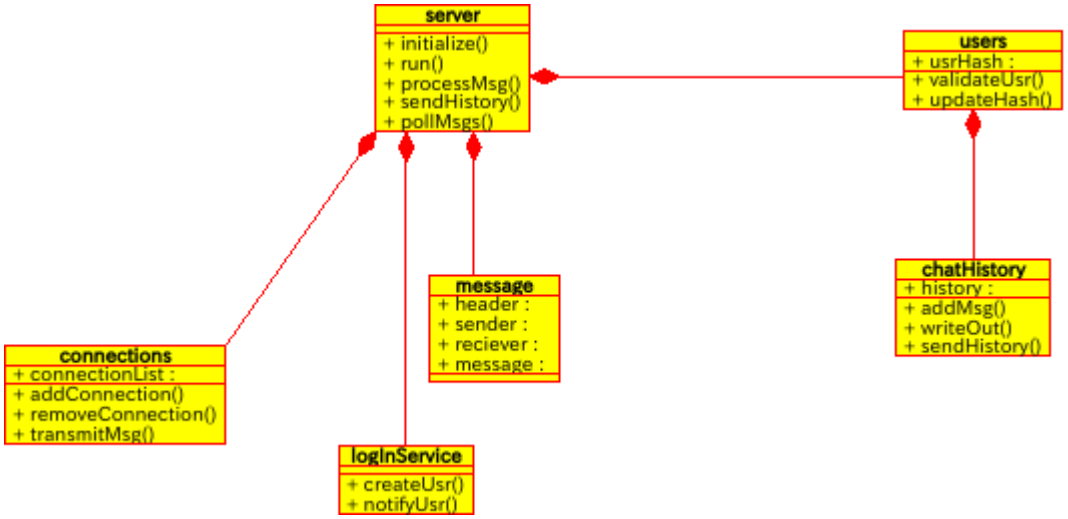
The overall design of this application is in two parts the server and the client. First let us look at the server on a high level. The job of the server is to be a central hub for information to pass to and from. When one user wants to talk to another they will send a message to the server and the server will decide who to send that message to. In this way the server controls the flow of the conversations. The server also needs to keep a history of the conversations so that when a user enters into the discussion they can see what has previously been said.

On the other side of this we have the client which makes up the second half of the application. The client program is the users means of interacting with the server and through that with the other users. The client program provides a simple means for a user to login and see and send messages, the below diagrams illustrate how the user interacts with the client and how the client interacts with the server.





The diagrams illustrate the case of first the user either logging in or creating an account and it shows how the information goes through a validation step and if that passes the server allows the client to connect and communicate with it, but if not the connection is rejected. The second diagram is an illustration of the flow of a message as it moves from user to user passing through the server. The details of the program design and class structure will be saved for the latter section on system architecture but below is the UML diagrams for the client and server.



As you can see from the diagrams the structure of the program relies heavily on composition of objects.

4 System Architecture

The structure of both the client and server rely on the composition of smaller classes to make a cohesive unit. These smaller pieces will communicate to each other by messages just and the users communicate to each other, the difference in these communications will be in the header of the message. Below is a break down of these systems and how they work.

4.1 Server

The server object is the overarching object on the server side its job is to manage all of the smaller objects in order to make them act as a cohesive whole. The functions that the server class has are simple initialize, run, processMsg, sendHistory, and pollMsgs. Initialize will initialize all of the smaller objects that make up the server. Run will be a while loop that keep the application open. The processMsg function exists to process the existing system messages and send them to the correct places this will be called from withing pollMsgs to create a message pump that handles the messages with a while messages processMsg loop. SendHistory simply grabs the history that is associated with each user and sends it to them so that they can see their previous conversation.
The only public method of the server class is initialize, run is called at the end of initialize, this is to ensure that all sub-systems are initialized before the server starts running and in running we pollMsgs which in turn processMsg and calls sendHistory if it was requested by a message.

4.1.1 loginService

The loginService object is responsible for creating new users and notifying the users object to update its hash if a new user is added. It also notifies the users object if a user tries to login so that users can attempt to validate the login. Like all of the components in this design the loginService communicate via message. It receives a message that has a header of either loginAttempt or newUser and then acts accordingly and creates a message to send to the users object with a header of validate or update depending on what the user object needs to do.

4.1.2 Users

The users object holds the usrHash which is a hash table of all of the registered users and their passwords. When a user tries to connect to the system they need to be validated by the users object. The user object has two functions validateUsr and updateHash. If the loginService passes a message of the type newUser the users object will updateHash which will add the user to the hash table if they do not already exist. If they do exist the users object will create a message that has a failed to connect message in it and pass that to the connections object that is in charge of picking up connections and transmitting messages so that the user can see that they failed and the client program can allow them to try again if they succeed then a success message will be sent to connections to let the client program and the user know. If the request from the loginService is for validation then the users object will attempt to look up the user and see if the password is the same. If it all checks out a sucessful login message will be generated and passed to connections to notify the client and user.

4.1.3 Connections

The connections object is the object responsible for controlling the flow of information into and out of the server. It has a list of connections, which is clients that the server is currently connected to and has three main functions addConnection, removeConnection, and transmitMsg. After a login is validated the connection needs to be added to the list and this is what add connection does. When a client disconnects from the server the removeConnection function removes that connection from the list. Finally when a Msg

needs to be transmitted either to a single user or the whole group the transmitMsg function is called and passed a message to pass along.

4.1.4 Message

The message is a struct that is used by all of the parts of the server to talk to each other it has 4 main parts the header, sender, receiver, and message. The header gives information about the type of message being sent for example if it is a loginAttempt or a personalMsg this tells the system that receives the message what to do with it. The sender is who sent the message, is it a message from a user to a user or from the loginService to the Users object. The reciever field is who the message is to again this could be a user or internal service. The final field is the actual message which in the case of a user talking to another or group would be the text they type in and in the case of a system call like loginAttempt it would be what credentials they are trying to pass to login.

4.1.5 ChatHistory

The chat history object is stored with the users in the userHash because it is a per user history of both the group conversation and private conversations they have been having. When a client logs in the sendHistory function is called which writes their history out to the client so that the user will have a chat session with all of the group chat and personal chat already in it. When a user sends a message through the client to the server the addMsg adds a message to the histories of the sender and reciever. When the client disconnects from the server the writeOut function is called that writes the history to a file so that it can be stored long term.

4.2 Client

The client object is the program that allows for the user to interact with the server and other users. The client itself is very simple in design and acts as a manager of other classes. The first thing the client does is initialize the other objects that you can see in the diagram that are composed with the client, it does this with a call to the function initialize. Initialize is the only public function of client initialize ends by calling run. Run

will have a while running loop that pumps the messages being created by the other objects to where they need to go.

4.2.1 InputHandler

The InputHandler object as the name implies handles input from the user. It calls sendMessage to send a message the user has typed in to all of the clients through the server. Internally it does this by creating a message where the receiver is all. Or it can create and send a personal message to a single other client by putting that client as the receiver when it calls the function personalMessage.

4.2.2 gui

The gui object is responsible for rendering the graphical user interface. When the state of the client changes by for example getting a new message it is passed to the gui objects render function which updates the screen with the new information. This is a very simple object with a very distinct purpose.

4.2.3 Usrs

The Usrs object is responsible for a local copy of all of the usernames of the people currently online. When a new user logs into the server through their client the server then broadcasts a message out which this object picks up and calls updateUsrs on to update the list of users that it holds, this function in turn generates a message for the gui to pick up and call render on to display the updated list so that the user can see who is online.

4.2.4 message

The message struct is the same struct that the server uses for the same purpose which is to allow the components of the client to communicate with each other and with the server.

4.2.5 login

The login object has two jobs to do, firstly when the client starts the loginPrompt function presents the user with a splash screen where they need to make an account or login with

valid credentials to use the application. When they do either of these things the second job of the login object is invoked which is in the validate function the validate function sends a validation or account creation request to the server to handle and then waits for a response and upon success closes the splash screen and allows the user to use the application.

5 Detailed System Design

Below is a the detailed design of each system and sub-system.

5.1 Server

Module name:	server
Module type:	object
Return type:	none
Input arguments:	none
Output arguments:	message
Error messages:	initializeFailed, runningStop
Files accessed:	none
Files changed:	none
Modules called:	loginService, Users, Connections, Message, ChatHistory
Narrative:	This module is the central management unit for the server program and as such has little input or output, it does however rely on a singly linked list of messages as a way to pass arguments to the modules that it calls this function runs a continuous while loop that says while there are messages read the header and pass them to the module that cares about that message.

5.1.1 loginService

Module name:	loginService
Module type:	object
Return type:	none
Input arguments:	message
Output arguments:	message
Error messages:	creationFailed, notValid
Files accessed:	none
Files changed:	none

Modules called:	Users
Narrative:	This module is responsible for receiving the messages that pertain to a login attempt or a login creation attempt. This module consumes these messages and creates a new message that it sends the Users object requesting either a lookup or an add of a user.

5.1.2 Users

Module name:	Users
Module type:	object
Return type:	none
Input arguments:	message
Output arguments:	message
Error messages:	validateFail, lookupFail
Files accessed:	userHash
Files changed:	userHash
Modules called:	Connections
Narrative:	This is where the hash table and file the hash table is written out to and is manipulated. This function compares the username in the sender field of the message it gets to the hash and the message field of the message to the password. If the header is Validate then this object just accesses the hash and compairs it data with the hash, but if the header is Create then it first checks the make sure the user does not already exists and if not modifies the hash and updates the file.

5.1.3 Connections

Module name:	Connections
Module type:	object
Return type:	none
Input arguments:	message
Output arguments:	message
Error messages:	ConnectionFailed

Files accessed:	none
Files changed:	none
Modules called:	none
Narrative:	This object maintains the linked list of connections that are currently active and sends and receives messages with them. Although this module does not call other modules it does communicate with them by outputting messages that they can then pick up. It is important to note that since multiple modules may need access to the messages that connections receives from clients all messages received from clients are pushed out to the system and then in the event that connections needs to send them on it picks them back up from the list that server is holding.

5.1.4 Message

Module name:	message
Module type:	struct(in java object containing only public fields)
Return type:	none
Input arguments:	none
Output arguments:	none
Error messages:	none
Files accessed:	none
Files changed:	none
Modules called:	none
Narrative:	This object is just data that the other objects use to communicate and act on which is why I treat it as a struct.

5.1.5 ChatHistory

Module name:	ChatHistory
Module type:	object
Return type:	none
Input arguments:	message
Output arguments:	message
Error messages:	HistoryUpdateFailed

Files accessed:	history
Files changed:	history
Modules called:	none
Narrative:	This module is responsible to maintaining and updating the history files that contain all of the conversations that people have had. It also sends this history as a message out for the server to pick up upon request so that clients can see the conversations from before they logged in.

5.2 Client

Module name:	Client
Module type:	object
Return type:	none
Input arguments:	messages
Output arguments:	messages
Error messages:	InitializationFailed, RunningStopped
Files accessed:	none
Files changed:	none
Modules called:	InputHandler, gui, Usrs, login
Narrative:	This object is the manager for the client program and is designed to initialize and then pump messages to the other systems that make up the client program.

5.2.1 InputHandler

Module name:	InputHandler
Module type:	object
Return type:	none
Input arguments:	input stream
Output arguments:	messages
Error messages:	NoIO
Files accessed:	none
Files changed:	none

Modules called:	none
Narrative:	This object is responsible for grabbing user input of f of the input stream and turning it into messages that the rest of the system can use.

5.2.2 gui

Module name:	gui
Module type:	object
Return type:	none
Input arguments:	message
Output arguments:	none
Error messages:	GuiFailed
Files accessed:	none
Files changed:	none
Modules called:	none
Narrative:	This object renders the user interface and updates it according to the messages that it receives.

5.2.3 Usrs

Module name:	Usrs
Module type:	object
Return type:	none
Input arguments:	messages
Output arguments:	messages
Error messages:	UpdateFailed
Files accessed:	users
Files changed:	users
Modules called:	none
Narrative:	This object takes in messages and updates the users list which is a file containing all of the users that are currently online, it then generates a message of its own to tell the gui to update the names it is to display. Again this communication is indirect as the object simply sends the message out to for the client to add to the list.

5.2.4 message

Module name:	message
Module type:	struct(in Java an object with only public fields)
Return type:	none
Input arguments:	none
Output arguments:	none
Error messages:	none
Files accessed:	none
Files changed:	none
Modules called:	none
Narrative:	This is the same as the one used by the server

5.2.5 login

Module name:	login
Module type:	object
Return type:	none
Input arguments:	message
Output arguments:	message
Error messages:	none
Files accessed:	none
Files changed:	none
Modules called:	none
Narrative:	This object takes the message for login that is produced by the input handler when a user tries to log in. It then creates a loginAttempt message or a loginCreate message and passes it along.