OOP Project - Chat Agent

-

Report

Matthieu DELOFFRE
Gabrielle GARNIER

4IR

February 5, 2020

Contents

1 User Manual

1.1 Starting the Chat Agent application

To set up the application for the first time, you need to extract the ChatAgent_Release.zip (or .rar) file. Once this is done, there are two ways to start the application (you won't need to extract the file anymore):

1. Double click on the ChatAgent.jar file in the unzipped file, or
2. open a terminal in the unzipped file directory, and type in the following command: "java -jar ChatAgent.jar"

Please make sure you have a proper JRE/JDK set up on your compute, or else you will not be able to use the app. We do not take any responsibility in case your Software doesn't work anymore because you deleted some of the files in the folder. That being said, you have the possibility to move the extracted file wherever you want in your directories.

1.2 Connecting to the system

When the application is launched, the first window you'll see is the connection window (*Figure 1.1*). You need to choose a username, and type it in the dedicated field. Then, click on the "Connect" button, or press the enter key. If the username you have chosen is available (i.e., has not been taken by another online user yet), you will connect to the system and get to the main window (see 1.3).

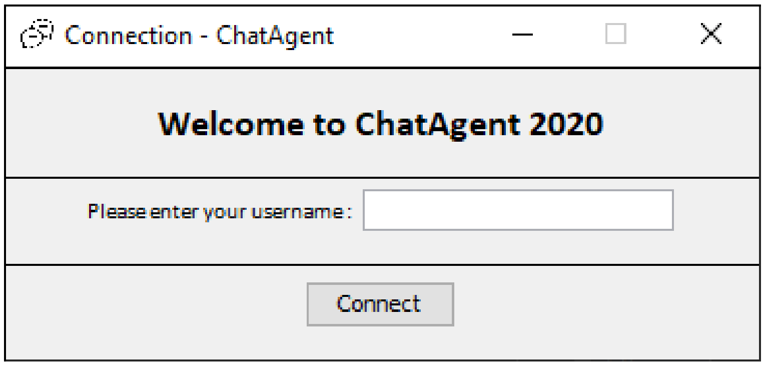
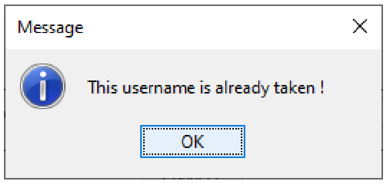


Figure 1.1: Connection window, with the username field, and a "Connect" button.

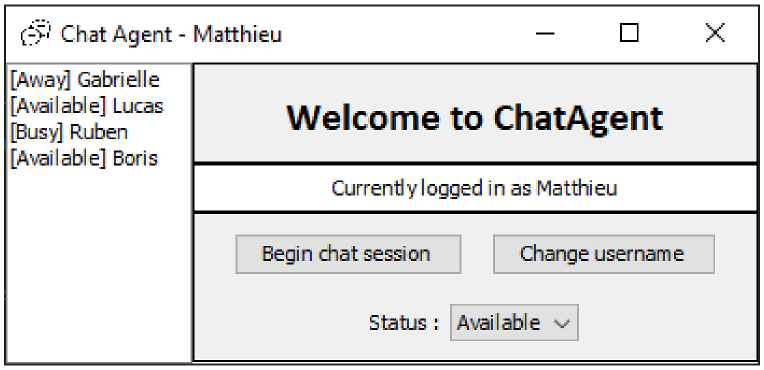
In case your desired username is not available, a dialog window will let you know. Click on the "OK" button, or close the dialog window, and choose another username.



This username is already taken !

Figure 1.2: If the chosen username is not available, this info window will appear

1.3 The main window


Chat Agent - Matthieu

[Away] Gabrielle
[Available] Lucas
[Busy] Ruben
[Available] Boris

Welcome to ChatAgent

Currently logged in as Matthieu

Status : Available

Figure 1.3: The main window of out Chat Agent

The main window is composed of a list of connected users (left side of the window), and the possible actions you may take (right side of the window). In the user list, every username is preceded of their current status. On the right side of the window, there are all the different actions you can take, like change your username, change your current status, or begin a chat session.

1.4 Changing the username

To change your username, you have to click on the "Change username" button on the main window (see *Figure 1.3*). A window similar to the connection one will appear, letting you type in your new username. Press the Enter Key or click on the "Change username" button to validate your choice.
Once again, if it is not available, you'll get the same dialog window as the one from *Figure 1.2*.

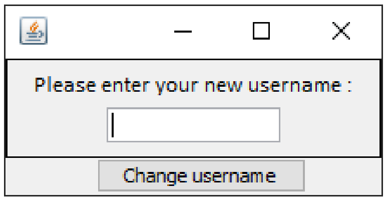

Please enter your new username :

Figure 1.4: Change your current username by typing it in this field

1.5 Changing your status

To change your status, you have to choose it from the combo box beneath the "Status:" label on the bottom right corner of the main Window (see *Figure 1.3*). Just click on the status you wish to have and it's done ! Once a status has been selected, every other active user will be notified of the change. A status can be changed at any time.

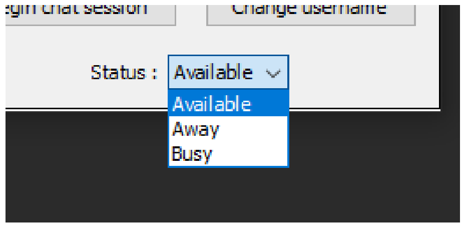

Status : Available

Figure 1.5: You can select your current status in the combo box

1.6 Starting a chat session with another user

To start a chat session with another user, you need to click on a username in the user list, and then click on the "Begin session" button. The user you selected must not have a "Busy" status in order to start a session with them. If no user has been selected, a dialog window will appear, asking you to choose a user in the list before clicking the button. Click "OK", then select a user in the list. Once a session has been started, the following window will appear.

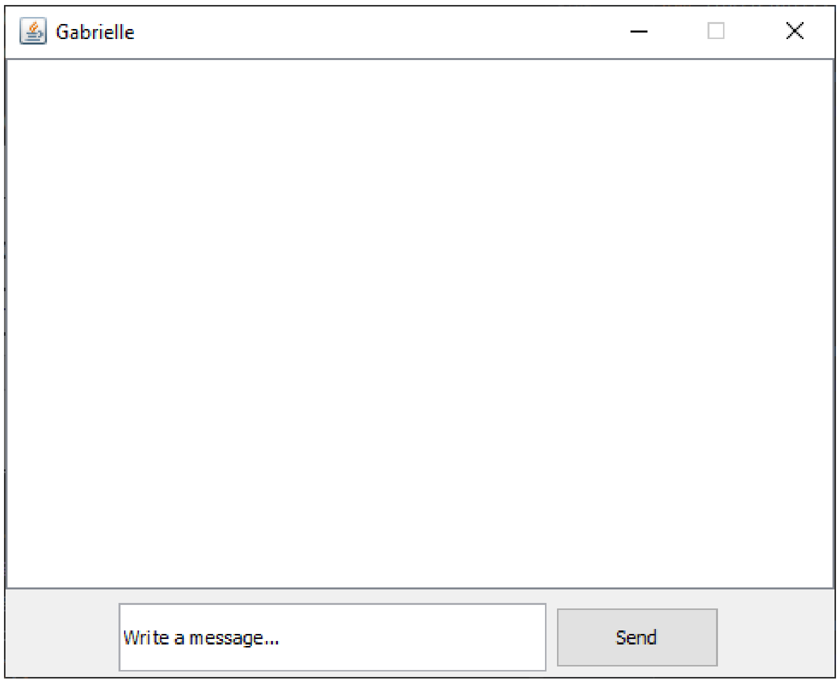


Figure 1.6: The basic chat window, without any message

The user you started a session with will only see the chat window if you send him a message. Sometimes, you will not be the user initiating the chat session. In this case, as mentioned right above, a chat window will appear on your screen as soon as someone sends you a message.

1.6.1 Sending a text message

In order to send a text message, simply type the desired message in the entry field, below the panel displaying the conversation. Then click on the "Send" button, or press the Enter key. Your message will be sent to the other user, and displayed right above on the conversation panel (with a blue border). If the other user sends a message to you, it will also be displayed in the conversation panel (with a red border). *Figure 1.7* shows you the chat window when some message have been sent and received.

Every message you send will be saved and displayed every time you will reopen the chat session with that same user. For the *Figure 1.6*, we started a chat session with a user we had never talked to via the Chat Agent before. If you happen to chance the device you are using at work, every saved message will be lost and all your conversations will start anew.

You can hover your mouse over the messages in the chat window: the timestamp at which the message has been sent or received will appear.

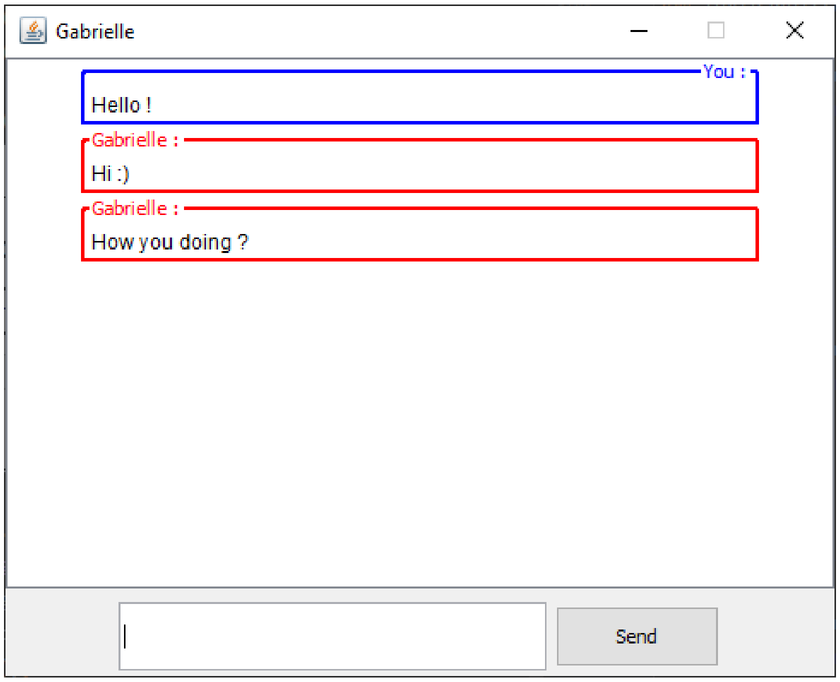


Figure 1.7: The chat window, once some messages have been sent

1.6.2 Sending an image or a file

To send an image or a file, use the drag and drop method: look up for the desired file or image in your computer directory, select then drag it onto the chat window. It will automatically be sent to the other user and displayed in the conversation panel (with a blue border).

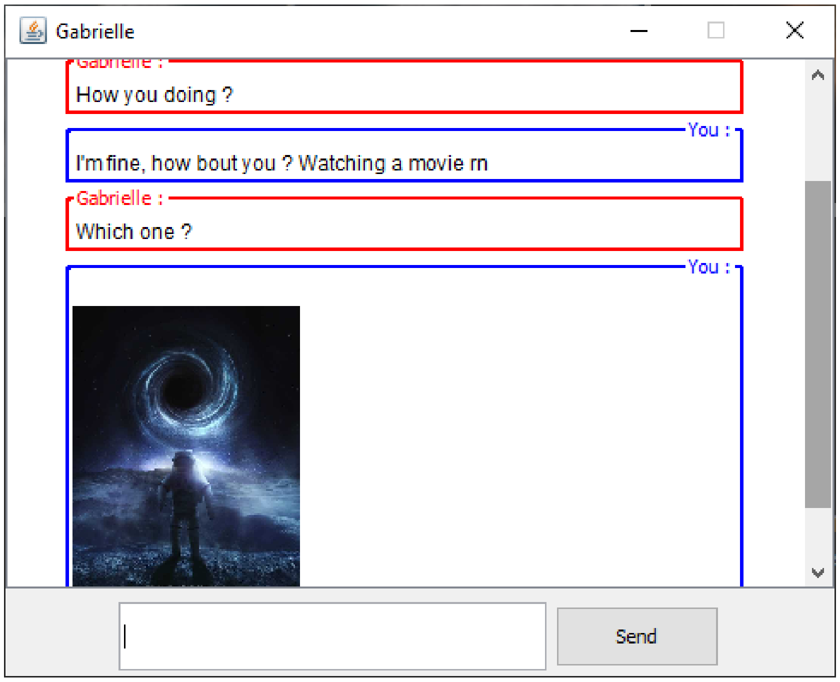


Figure 1.8: The chat window, when an image is sent

1.6.3 Receiving an image or a file

When a image or a file has been received, it is possible to save it to your computer. To do that, click on the file/image displayed in the conversation panel, and the following window will appear. From there, you can chose where you would like to save the image/file. It is recommended to do so, as non text-messages are not saved in the message history.

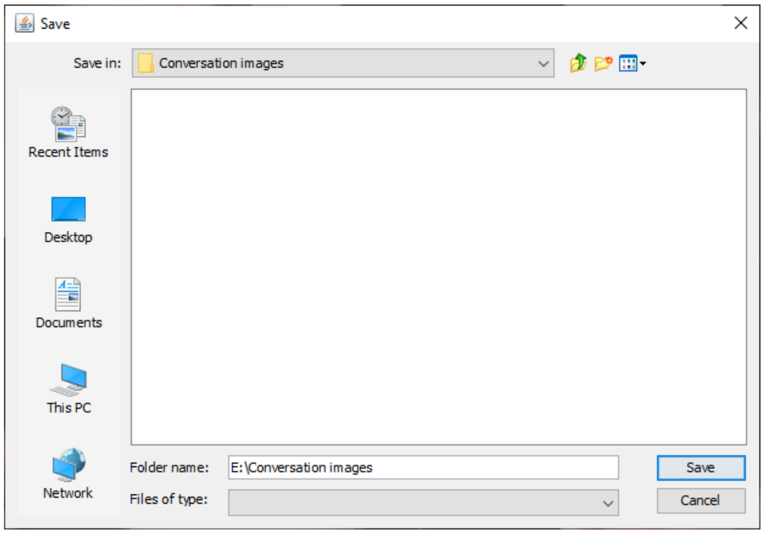
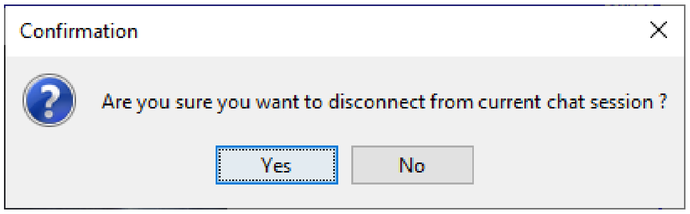


Figure 1.9: The window from which you have to chose the destination folder

1.7 Ending the chat session

To end a chat session, simply click on the red cross at the top right of the window to close it. A dialog box will appear, asking for confirmation. Click "Yes", and the session will be ended, for both you and the person you were talking to.



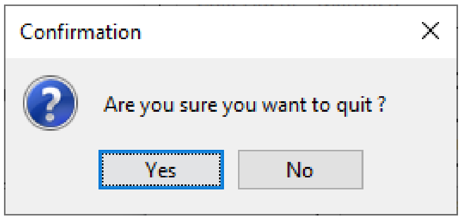
Confirmation

Are you sure you want to disconnect from current chat session ?

Figure 1.10: Disconnect confirmation for the chat session

1.8 Disconnecting from the Chat Agent

To disconnect from the system, simply close the main window. A confirmation window will appear, click "Yes" to quit the Chat Agent. You will disappear from the active user list on the main window of other users and they won't be able to message you anymore.



Confirmation

Are you sure you want to quit ?

Figure 1.11: Disconnect confirmation for the Chat Agent

2 Test procedures

This chapter is a tiny summary of how we tried to test every functionality of our software without any kind of JUnit test implemented (unitary tests are kind of hard to set up for what we had to test). Most of the results were deducted from prints on the command prompt, and java Exception not being caught. All of these tests have been handmade, with the simulation of between one and a hundred users logging onto the system.

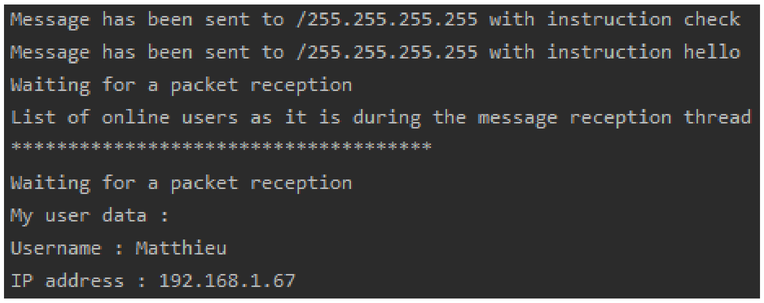
2.1 Testing the connection procedure

The connection procedure has been verified through the following tests:

- Connecting as the first user
- Connecting when other users are already connected
- Trying to connect with an unavailable username

2.1.1 Connecting as the first user

When connecting to the system as a first user, there is not much we can test. We just printed some info on the terminal in order to make sure that everything was going on smoothly. As every sensible line of code is surrounded by exceptions catchers, no exception in the terminal meant that the system worked for one single user connecting to the system.



```
Message has been sent to /255.255.255.255 with instruction check
Message has been sent to /255.255.255.255 with instruction hello
Waiting for a packet reception
List of online users as it is during the message reception thread
************************************
Waiting for a packet reception
My user data :
Username : Matthieu
IP address : 192.168.1.67
```

Figure 2.1: There are a few prints in the terminal telling us what's going on in the program

We can see the program properly going into waiting state, with an empty list of users, as no one else is online.

2.1.2 Connecting when other users are already connected

If we connect with other users already connected, we checked with various amounts of connected and connecting users that every active user list would update itself properly.

2.1.3 Trying to connect with an unavailable username

This test is fairly easy to do. We simply connected three users with different usernames in the system. Then, a fourth user tried to log on, using all three taken usernames. Every time, the error window showed up, telling us the username was not available. We then changed the username from an already connected user (from "user3" to "user3.1"), and tried to connect the fourth user with *(1)* the newly changed username "user3.1" -the dialog box showed up once again, and *(2)* with the previous username "user3". This time, we were able to log onto the system. We then deduced that our authentication procedure was working as intended.

2.2 Testing the username change

The first tests for the username change were quite simple. We simply tried to change usernames while 0, 1, or more other users were connected. Every name change worked perfectly. Further tests for the username change have been done with opened chat session, they are described in the next section.

2.3 Testing the actual chat session

We automated some tests for the chat sessions tests, with a hundred fictive users connected on the system. Approximately 200 chat sessions were opened at the same time, with automatic messages being sent in every session about once every 5 seconds. Everything went on smoothly without the program catching any sort of exception. We then tried to change the usernames of some of these users while having the sessions opened, to check two different things: *(1)* The usernames in the main window's user list are actualized and *(2)* The usernames in the chat session are also actualized, and the sessions do not end. After a manual verification, we confirmed every username had been updated. The same tests have been made for the status change, and the result was also positive.

2.4 Other tests

The rest of the tests we made primarily consisted of checking if the conditions described in the specifications of the project, like for example:

- Our system does work on various operating systems (Windows, Linux). We could not test it on OS X though.
- The size of our system is way inferior to 50MO.
- As we are using the TCP Protocol, the integrity of our messages is higher than 99,999%.

Those tests have been done by hand with no automation, as it is only needed to check facts.

3 Other information

3.1 Code organization

There are a few important information about how we decided to implement our chat system. One of the main issue was how to save the message history, in order to display it later when the conversation is opened again. We chose to make the following hypothesis: assuming our software will be used in a company, everyone will use it from the same device every time -people usually don't swap their offices- and thus, we can store their message history directly on the computer, without the need to use an online database.
The format in which the message are saved is the .json, as it is fairly easy to use. There are lots of existing libraries for the JSON file treatment in java, and our choice ended up to be the json-simple library, which, as the name suggests, gives simple methods to deal with JSON files. A file containing the message history is named after the MAC address of the other user (without the ':', as windows does not accept this character in file names), and is stored in a "conversationData" file next to the executable jar file. This assumption also brings some simplification when it comes to the user identification. Since the computer remains the same every time, we can easily identify everyone in the system through their MAC address.

For information about the code itself, we decided to split it into three different parts, which can be described like this :

- The interface part
- The software part
- And a last part entirely dedicated to the chat session (front-end & back-end)

The reason we ended up organizing it this way is that the session needed a very close link between interface and software management (closer than the other classes). It's an approach close the the MVC design pattern -we have the model with the software part and the view with the interface part-, but it's still unique.

As for how the software actually do things, it would be easier to clarify the question with a class diagram, as well as some sequence diagrams for some use cases.

3.1.1 Class diagram

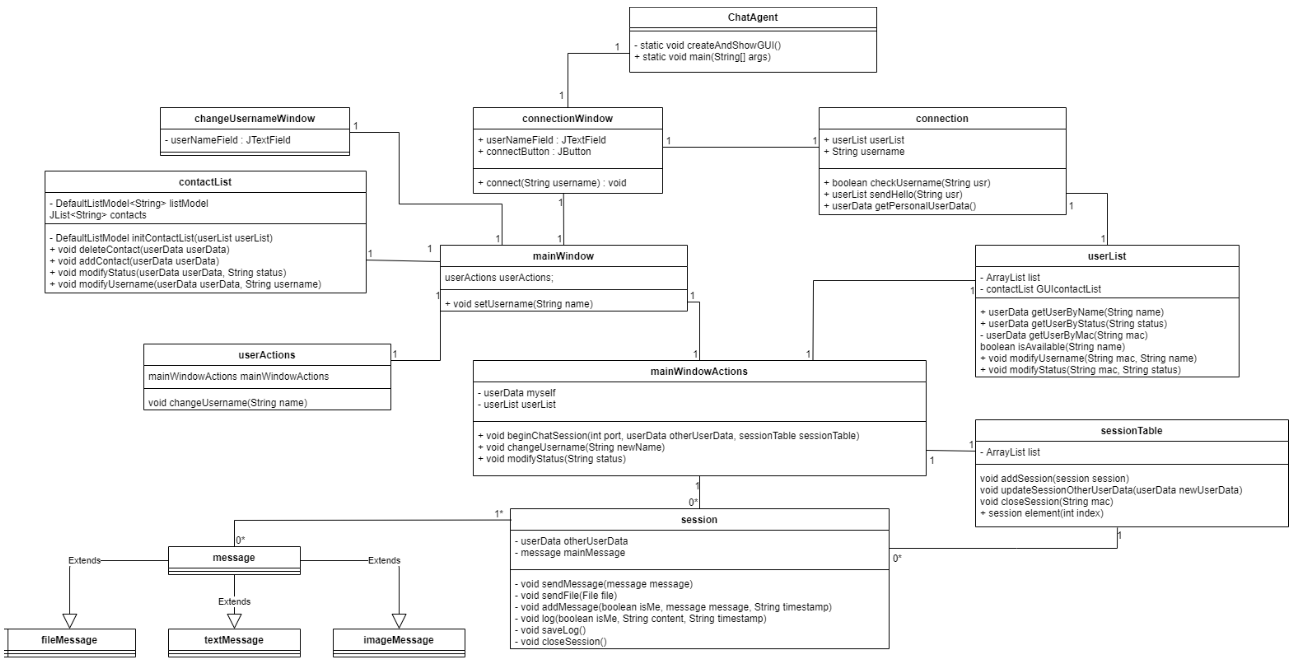

ChatAgent
- static void createAndShowGUI()
+ static void main(String[] args)
changeUsernameWindow
- userNameField : JTextField
connectionWindow
+ userNameField : JTextField
+ connectButton : JButton
+ connect(String username) : void
connection
+ userList userList
+ String username
+ boolean checkUsername(String usr)
+ userList sendHello(String usr)
+ userData getPersonalUserData()
contactList
- DefaultListModel<String> listModel
JList<String> contacts
- DefaultListModel initContactList(userList userList)
+ void deleteContact(userData userData)
+ void addContact(userData userData)
+ void modifyStatus(userData userData, String status)
+ void modifyUsername(userData userData, String username)
mainWindow
userActions userActions;
+ void setUsername(String name)
userList
- ArrayList list
- contactList GUIcontactList
+ userData getUserByName(String name)
+ userData getUserByStatus(String status)
- userData getUserByMac(String mac)
boolean isAvailable(String name)
+ void modifyUsername(String mac, String name)
+ void modifyStatus(String mac, String status)
userActions
mainWindowActions mainWindowActions
void changeUsername(String name)
mainWindowActions
- userData myself
- userList userList
+ void beginChatSession(int port, userData otherUserData, sessionTable sessionTable)
+ void changeUsername(String newName)
+ void modifyStatus(String status)
sessionTable
- ArrayList list
void addSession(session session)
void updateSessionOtherUserData(userData newUserData)
void closeSession(String mac)
+ session element(int index)
session
- userData otherUserData
- message mainMessage
- void sendMessage(message message)
- void sendFile(File file)
- void addMessage(boolean isMe, message message, String timestamp)
- void log(boolean isMe, String content, String timestamp)
- void saveLog()
- void closeSession()
message
Extends
Extends
Extends
fileMessage
textMessage
imageMessage
1
1*
0*

Figure 3.1: The class diagram for the project

3.1.2 Sequence diagrams

3.2 Things to be improved

There are a few things that could have been done a bit better. First of all, there is the interface. It's the first thing the user sees, and gives a first impression about the application. With a bit more time and experience with GUI (it was the first time we had to implement an interface on our own, from scratch), it would have been possible to display a cleaner, smoother interface, with better suitability for every kind of operating system.

One of the main thing to improve would be the way we identify the users on the system (through the MAC address). There must certainly be a better way to tell people apart, that doesn't require people to stay on the same device. That being said, it's not impossible to change computer, it will just make you lose all the chat history with everyone (and everyone who discussed with you before will lose it, too).

Another path we could take to improve our software would be replacing the JSON chat history system with a real database. The identification method would stay the same but all the conversation data would be safe on the cloud, immune to some wrong move coming from the user.

Lastly, there is the servlet implementation. We managed to properly send the updates to our tomcat server every time something is going on on the Chat Agent (like someone connecting to the system, changing their username, changing their status, etc), but we couldn't make our system receive the updates nicely, so we decided not to ruin the entire program by making it rely only on the servlet. That means the only part about the servlet left in our code is the lines sending the updates to the server. One clear way of improving this is to finish this implementation and make our program receive and treat the updates from the server.