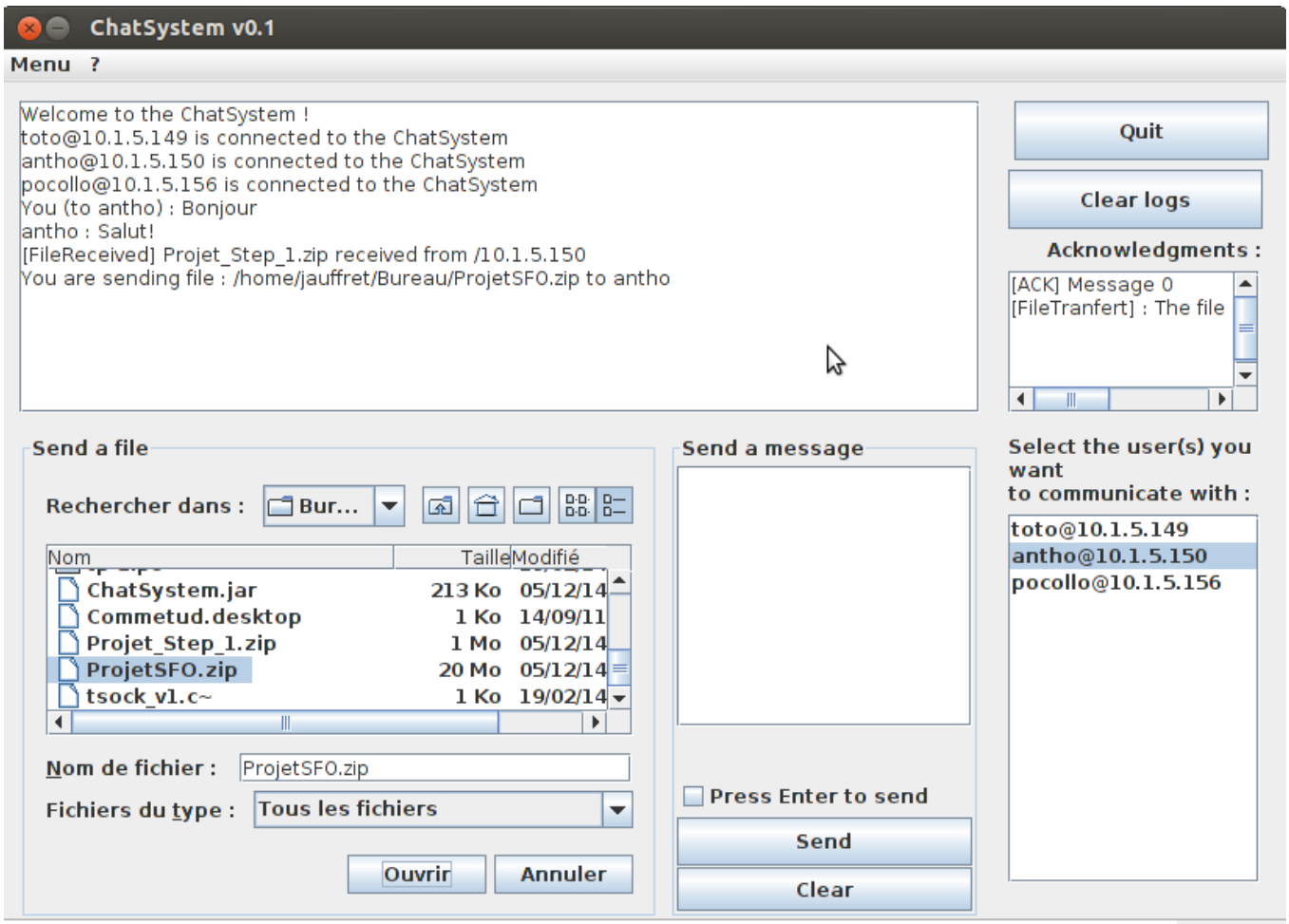
ChatSystem Report

Testing our application

Throughout the implementation phase, we did a lot of tests in order to be sure that our application will be able to realize the four basic use cases that we found in the SRS : connect, disconnect, send and receive message and files.

We were able to test the interaction of our application with the ones from the other groups and they all worked fine together.

We made a few screenshot of the way our application works. You will see it realizing every basic use cases, followed by the trace made in the console explaining each step of the program:



On this screenshot, you can see that we realized every basic actions of our SRS : connect, see other users connected, receive a message and a file, send a message and a file.

This is a screenshot from the first window asking you to enter your username and then connecting.

Here is the trace of the execution in the console :

This is the part when we connect and we send our Hello, and retrieve the hello acks.

```
jauffret@geitp111-5:~/Bureau$ java -jar ChatSystem.jar
TCPServer : Creating server socket on port 13
TCP Server : Waiting for a TCP request
UDPSender : UDP Socket opened on port 1337
UDPReveiver : Ready to receive
GUI : connectButtonPushed
Controller : Send Hello (broadcast) with nickname = toto
NI/SendHello -> Message to be sent : {"messageNumber":-
1,"userName":"toto","messageData":"","type":"hello"} to 255.255.255.255
 UDPSender : Packet sent through the port 1337
UDPSender : Packet = java.net.DatagramPacket@49f6d04a
UDPReveiver : Packet received : java.net.DatagramPacket@5f81c977
NI/HandlePacket : DatagramPacket received
NI/HandlePacket : Received packet type is hello
Controller : HelloReceived from toto@10.1.5.149
Controller : We add toto@10.1.5.149 to the user list
GUI : setTextLog ( toto@10.1.5.149 is connected to the ChatSystem )
NI/SendHelloAck -> Message to be sent : {"messageNumber":-
1,"userName":"toto","messageData":"","type":"helloAck"} to 10.1.5.149
UDPSender : Packet sent through the port 1337
UDPSender : Packet = java.net.DatagramPacket@c63679a
UDPReveiver : Ready to receive
UDPReveiver : Packet received : java.net.DatagramPacket@5226b5d0
NI/HandlePacket : DatagramPacket received
NI/HandlePacket : Received packet type is helloAck
Controller : HelloAckReceived from antho@10.1.5.150
Controller : We add antho@10.1.5.150 to the user list
GUI : setTextLog ( antho@10.1.5.150 is connected to the ChatSystem )
UDPReveiver : Ready to receive
```

Initialisation, opening the sockets and listening

Hello sent in broadcast

Receiving my own Hello

Sending HelloAck to me

Receiving HelloAck from a remote user

This is the trace when we exchanged messages (I sent a message to antho, and then received one from him) :

```
GUI : sendButtonPushed
Controller : Send Message to users selected
GUI : setTextLog ( You (to antho) : Bonjour )
NI : Ready to send "Bonjour" to toto @10.1.5.150
NI/sendMessage -> Message sent :
{"messageNumber":0,"userName":"toto","messageData":"Bonjour","type":
"message"} to 10.1.5.150
UDPSender : Packet sent through the port 1337
UDPSender : Packet = java.net.DatagramPacket@b47af37
UDPReveiver : Packet received : java.net.DatagramPacket@f4049aa
NI/HandlePacket : DatagramPacket received
NI : Received packet type is messageAck
Ack number is : 0
Controller : Ack n° 0 received
GUI : setAckLog( [ACK] Message 0 )
UDPReveiver : Ready to receive
UDPReveiver : Packet received : java.net.DatagramPacket@1308c2db
NI/HandlePacket : DatagramPacket received
NI/HandlePacket :  Received packet type is message
Controller : Message received from antho
GUI : setTextLog ( antho : Salut! )
NI : Ready to ack messageNumber "0"
NI/sendMessageAck -> Message sent : {"messageNumber":0,"userName":" "
,"messageData":" ","type":"messageAck"} to 10.1.5.150
UDPSender : Packet sent through the port 1337
UDPSender : Packet = java.net.DatagramPacket@2430c1a0
```

Sending a message to “antho”

Receiving MessageAck from “antho”

Receiving message from “antho”

Sending MessageAck to “antho”

And finally here is the trace when we exchanged files (I received a file from him, and then sent him one) :

```
TCP Server : Created a new TCP Receiver
TCP Server : Waiting for a TCP request
TCPReceiver : Ready to receive Projet_Step_1.zip , size : 1412805
TCPReceiver : Tranfert succeded ! Received : Projet_Step_1.zip
Controller : Inform GUI file received
GUI : setTextLog ( [FileReceived] Projet_Step_1.zip received from /10.1.5.150 )
GUI : Send File action request
Controller : Send File to users selected
GUI : setTextLog ( You are sending file : /home/
jauffret/Bureau/ProjetSFO.zip to antho )
NI: Sending : /home/jauffret/Bureau/ProjetSFO.zip to 10.1.5.150
TCPSender : creating socket with remoteIP = /10.1.5.150 port : 1337
TCPSender : success. Ready to send file.
TCPSender : File to be sent : ProjetSFO.zip
TCPSender : FileInputStream Succeed
TCPSender : Informations such as filename and filesize collected
TCPSender : Filename and filesize sent
TCPSender : File data sent
TCPSender : Transfert file has succeeded ! Close socket.
Controller : File Tranfert Result received, sending to GUI
```

Receiving file : "Projet_Step_1.zip " from "antho"

Sending file " ProjetSFO.zip" to "antho"

Organisation of the work

Concerning the organisation of our work, we had gathered throughout the conception with the rest of the group in order to determine the communication protocol that we were going to use. We had to agree on the use of JSONS, and on the JSONS keys. For the file transfer, we had to gather again, to determine how we were going to send the file name and size.

Concerning or work as a duo, we had to share the work in an efficient way. We worked using SVN, separating the tasks, working at home separately on a precise functionality and then putting it in common using versioning tools, and discussing it during the TP. We had to handle SVN conflicts, and really learn how to work in pair. (reading each other's code, taking risks, and evaluating the amount of work required for each tasks).

Changes in regard of the SDD

During the implementation phase, we found out that we had not a detailed enough vision of our program. We had not anticipated every single detail during the conception time.

Of course, we kept the global model of our application, using the general concepts of : MVP, boundaries/entities, the “façade” pattern using interfaces, and the separation in GUI,NI, and Controller.

But on multiple details we had to make some modifications.

Handling the messages

For example, we had not anticipated very well the way packets are handled. We had to implement an abstract factory in order to manage the Messages and JSONS.

Arguments in the functions

We had sometimes also not very well anticipated the arguments of our functions, for example with ip addresses, or we had sometimes overestimated the content of the messages in regard of the choices that we made with the rest of the group. (for example, in the beginning we thought that we would directly have the nickname when we get a message, but we had to retrieve it in the list, using the ip address.)

Managing the JList

We also had not thought about the fact that JList were not automatically updated, we handled this by creating a new class inherited from DefaultModelList that handle everything. (we could easily modify the way our User class work, it will be transparent for the controller who only uses the class UserList)

Signals

The names of the signal have slightly changed from the original conception. But globally, we managed to respect the original design, using the correct terms. And in our sequence diagram, we had not put the use of the “façade pattern”. No components can communicate directly with the controller, only the GUI and the NI can call methods from the controller.

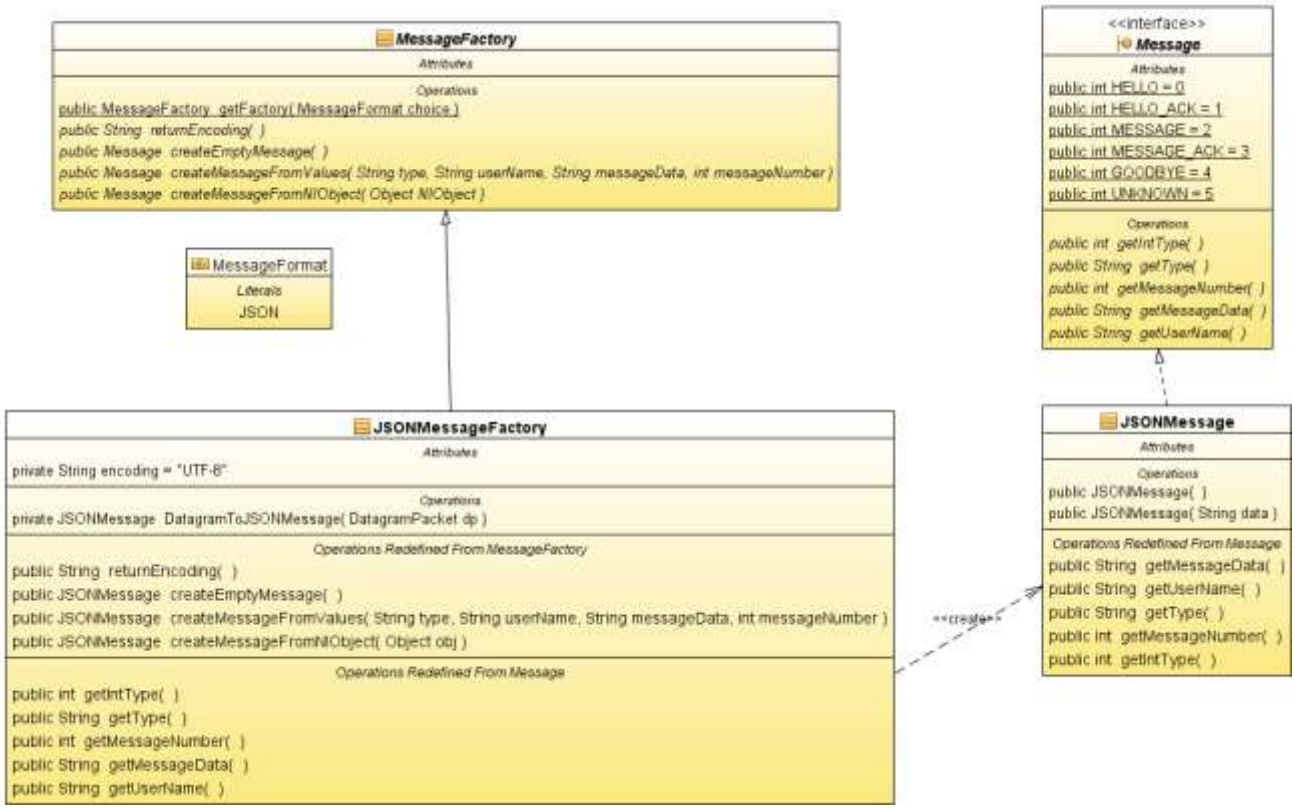
File transfer

The file transfer management of our application has been modified quite a lot. In fact, we were not really clear during the design phase on how the file transfert is made in Java, we did not know at this time every class involved in this functionality (the concept of stream, the way the socket works,etc). We really learned how they all work together by implementing them.

Generated UML Examples

We generated a few UML in order to see the modification, and really see what we have done in comparaison of what was planned.

Here is of example our abstract factory generating message :



You will find in our documentation, PSM Class diagrams, and the diagram for NI. We have not choosed to put them here because of their size. They will not be readable in a text document like this one.

The UML generation has some quite obvious limitations. We were never able to truly obtain the diagrams that we wanted, and the plugins were quite experimental (causing us to reinstall netbeans two times for example). But it can give us a really good overview of our program, and how classes are linked together.