INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

DOCUMENTATION ON THE CHAT APP

For the attention of Mr. Sami Yangui,

Paul Etse & Charan Nalakalasala

2019-2020

Contents

I. Introduction

This document describes the conception of our chat system project. The realization of the project is accessible from https://github.com/paulEtse/ProjetJava.

II. App details

The main objective of this app is to chat with multiple users by discovering the connected users.

1. Tools used to create the application

- We created interface with Java Swing
- We used JDBC connector for communication with the DB (Database). The application's data is stored in a remote database.
- We used http requests for the communication with Servlets.
- UDP socket is used to manage the Broadcast
- TCP Socket is used to manage the communication between users
- Maven is used to manage the project and its dependencies, so that we can have a standalone app that can be run on any machine using JVM (Java Virtual Machine) with the following command

2. How does it works

In the login page, you set your login and you will discover the connected users, to chat with users click on the desired user's name then you will get connected. The system must automatically retrieve the history of messages with whom you have connected in the panel.

3. Compilation

To compile the app and create an executable file, you can use the command:

- maven clean
- maven build
- maven assembly:single

To compile the servlet and create the war file, you can use the command:

- maven clean
- maven build
- maven install

4. Execution

To execute the app, in a windows system just double-click the chatApp.exe file. You can also use the terminal with the java command:

```
java -jar chatApp.jar
```

5. Test

You can test the app on two (or more) machines by following these steps:

- Install java in the machine
- Download the chatApp.jar file
- In your terminal, navigate to the folder that contains the downloaded chatApp.jar file and type the command java -jar chatApp.jar: the app will be launched in a new window, you can then choose your login. App will show connected users, you can select any of

them and start chatting with him. You will get the history of your old conversations with that user. You can also exchange files.

When a user launches the app, he will see connected users and will be asked to set his login, as shown in the figure

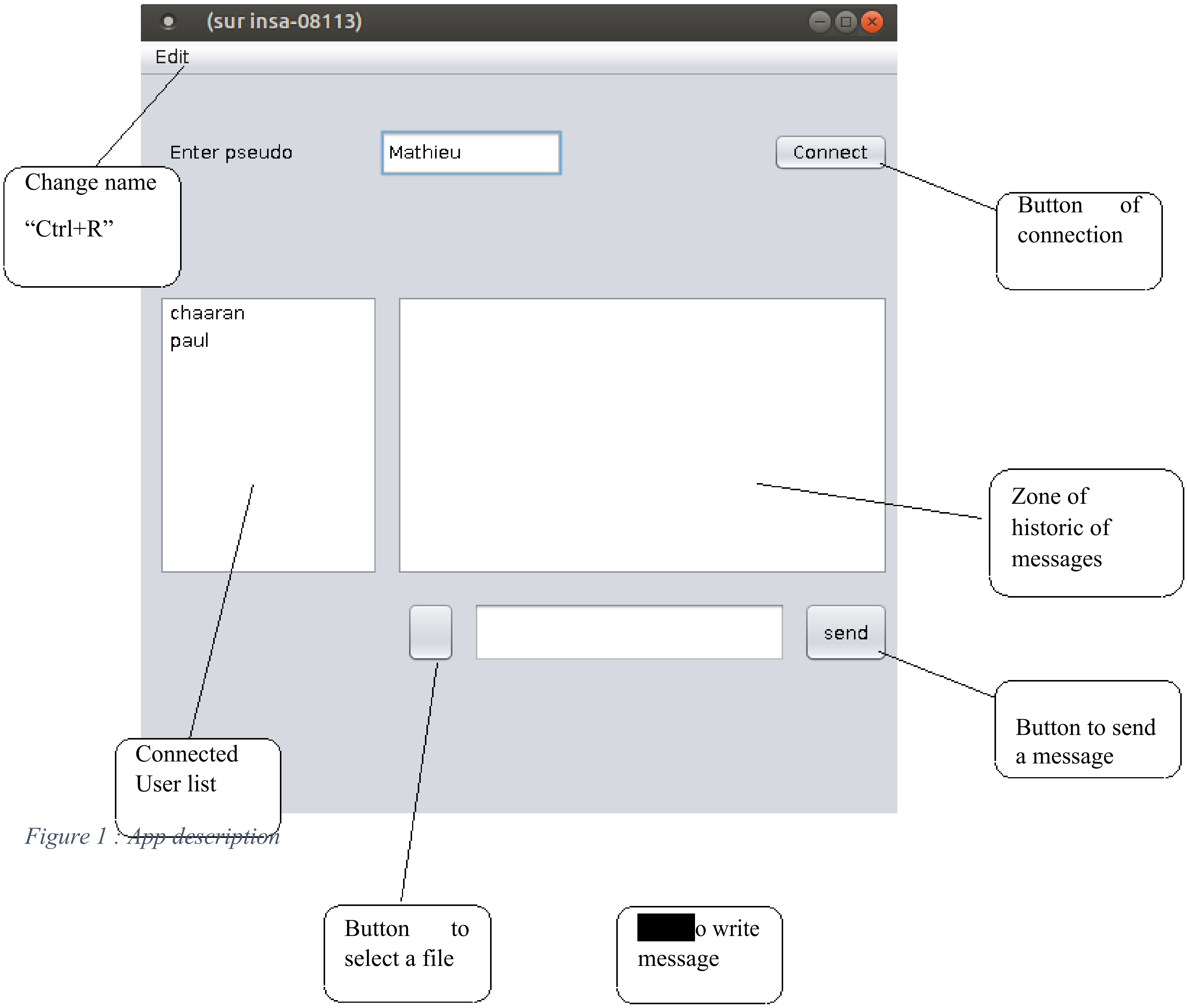
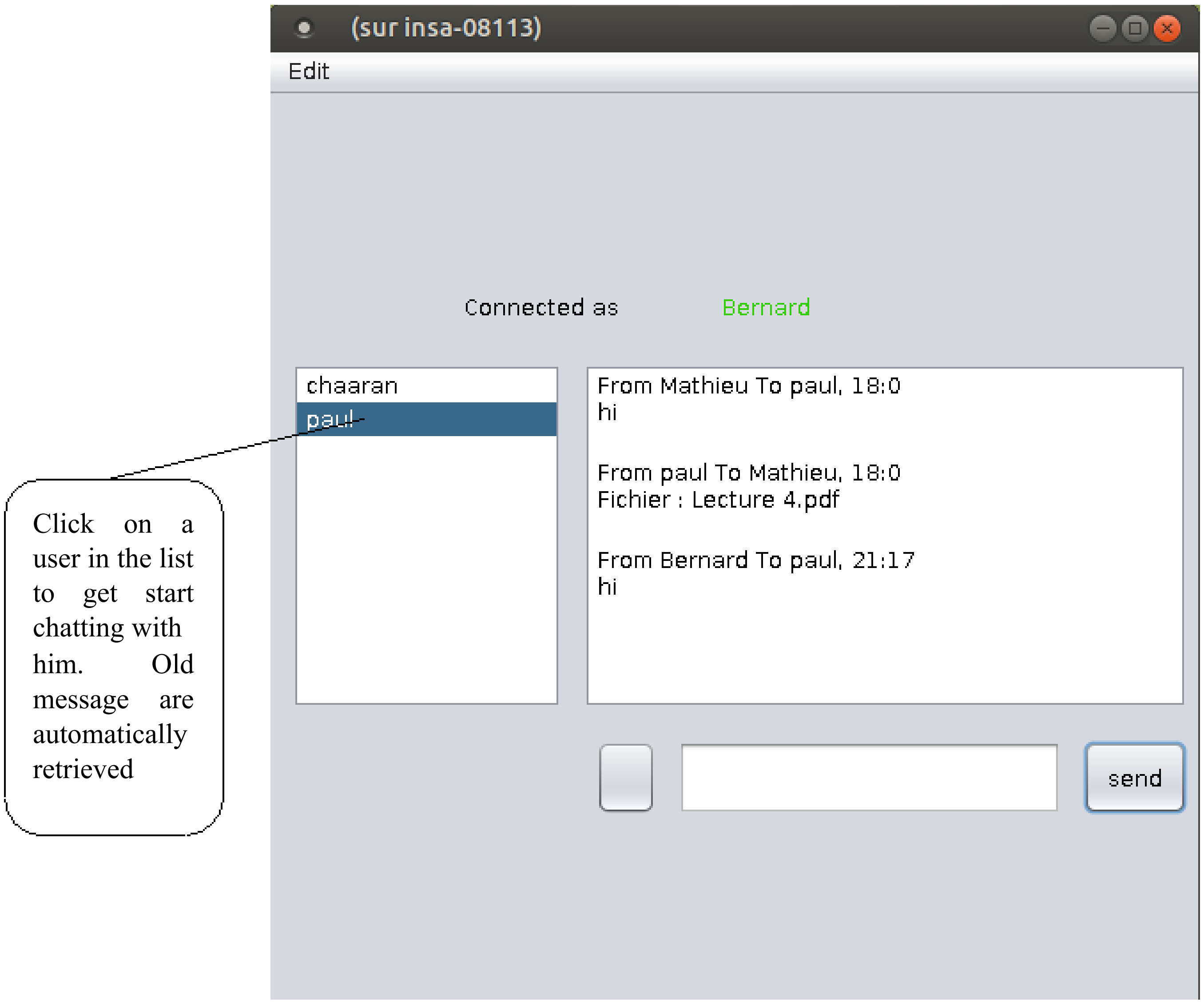


Figure 1 : App description

When a user is connected, he can click on any connected user login in the box at left, get the history and start a chat with him. He can also send files to that user.



From Mathieu To paul, 18:0
hi

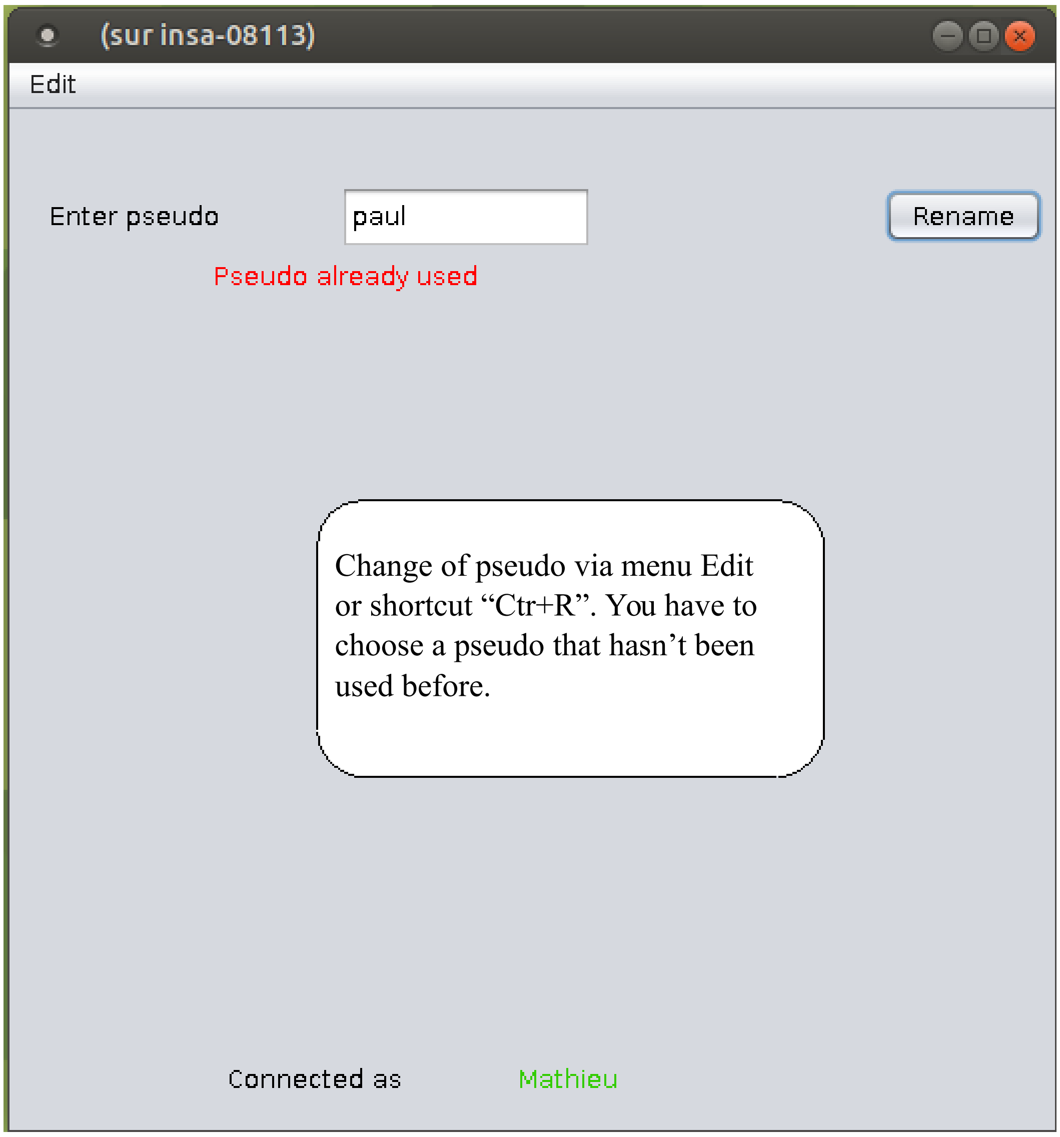
From paul To Mathieu, 18:0
Fichier : Lecture 4.pdf

From Bernard To paul, 21:17
hi

Click on a user in the list to get start chatting with him. Old message are automatically retrieved

Figure 2: chatting with a specific user in the app

A user can change his login. However, he must choose a name that is not already used. Otherwise, he will get an error message, as shown below.



Change of pseudo via menu Edit or shortcut “Ctr+R”. You have to choose a pseudo that hasn’t been used before.

Figure 3: Change name in the app

III. Database

We save all the messages that happen between sender and receiver with IP address as primary key. For connection, users send this info to the database. The sender saves every message sent in the database. This database helps in retrieving the old messages. Here is the structure of our database:

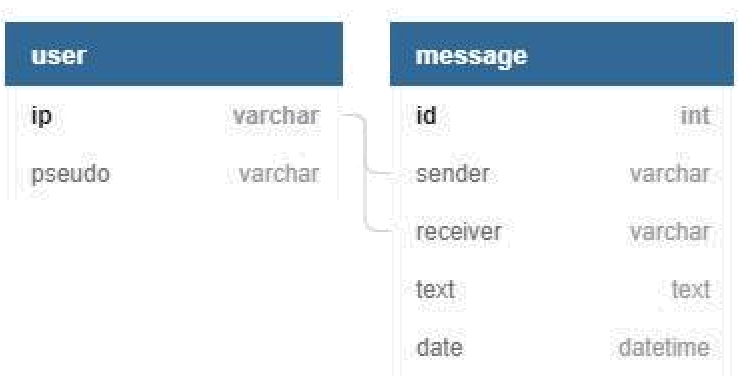


Figure 4: Relational database structure

We use mySql database server of INSA via the machine (srv-bdens.insa-toulouse.fr). You can change this in the configuration files by setting up your own database.

IV. Servlets

Java servlets, a standard for implementing Java classes that respond to requests. We used them here to allow remote users to communicate with local users. In fact, the app can allow a remote user to send messages to local users and vice-versa. Therefore, after the connection or disconnection, each user sends a message to the servlet. The servlet can then notify all remote users to update their connected user list.

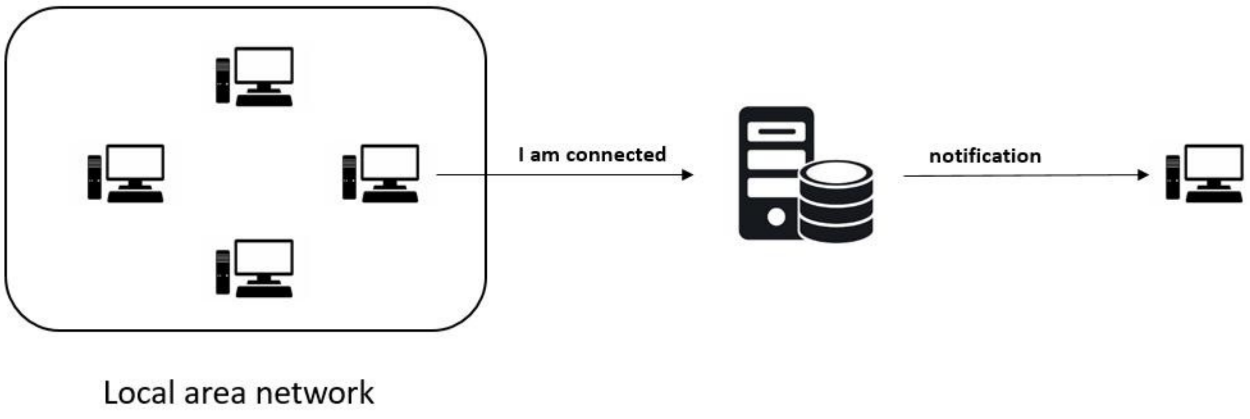


Figure 5: servlet

We use servlets that are deployed on the tomcat server of INSA (srv-gei-tomcat.insa-toulouse.fr). You can choose to use your own servlet by setting up your new servlet address in the Global.java file. You can also run the app without servlets. In this case just need to set the variable Subscription to false in the configuration files.

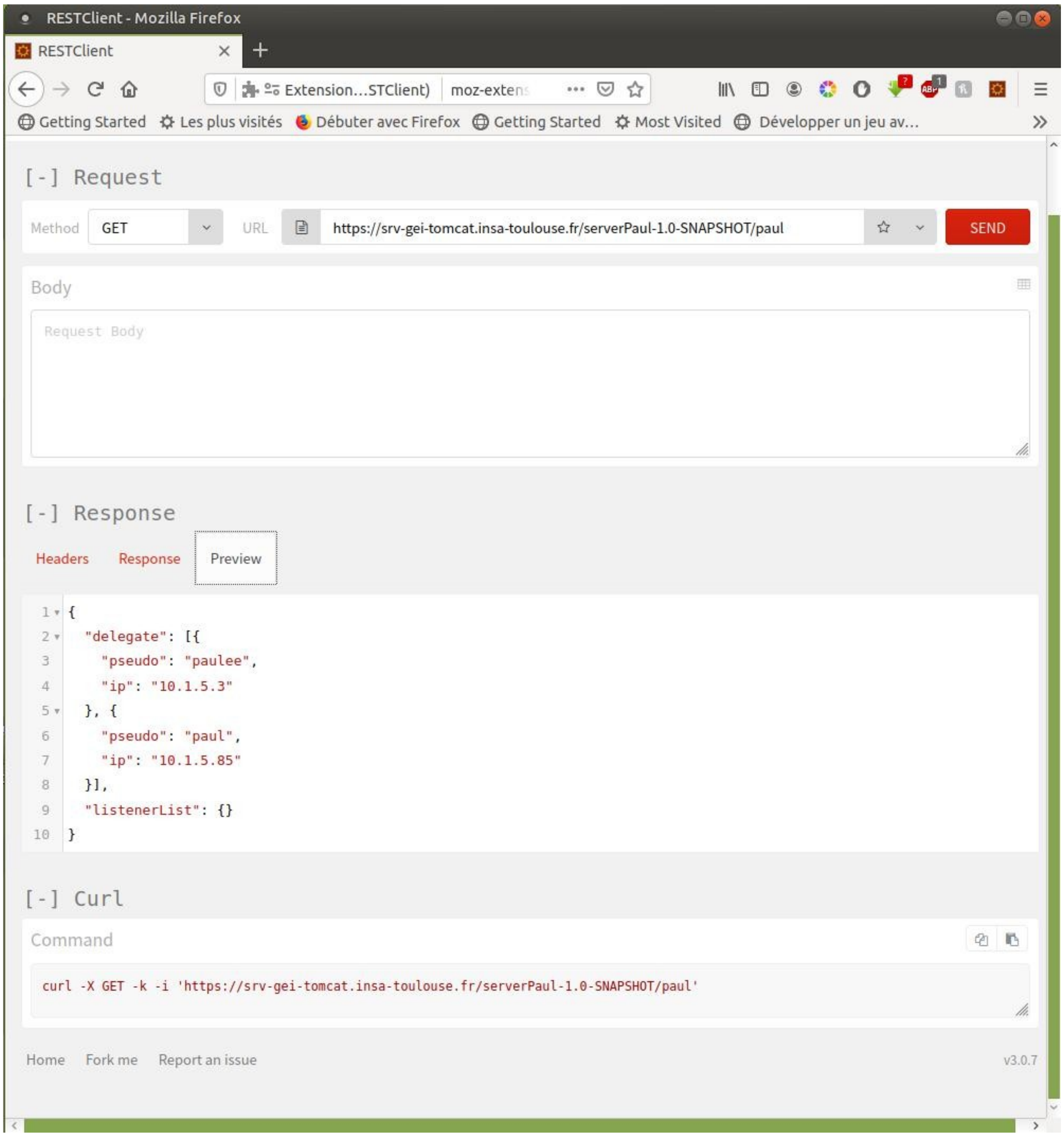
The exchange between the app and the servlet is in JSON format. The protocol http is used for communication.

To test our servlet we use a REST (Representational State Transfer) Client (Postman for example) to send our request to the servlet.

Our requests are:

GET: it return the connected user list
POST: it sends information's about new connected user
PUT: it is used to update information's of user (when he change his name for example)
DELETE: it is sent when a user is disconnected.

The following image show an example test of get request. It return two users.



Method: GET
URL: https://srv-gei-tomcat.insa-toulouse.fr/serverPaul-1.0-SNAPSHOT/paul

```
{
  "delegate": [{
    "pseudo": "paulee",
    "ip": "10.1.5.3"
  }, {
    "pseudo": "paul",
    "ip": "10.1.5.85"
  }],
  "listenerList": {}
}
```

```
curl -X GET -k -i 'https://srv-gei-tomcat.insa-toulouse.fr/serverPaul-1.0-SNAPSHOT/paul'
```

Figure 6: Test of servlets

V. UML diagrams

1. Use case Diagram

This diagram describes the main use cases of our system.

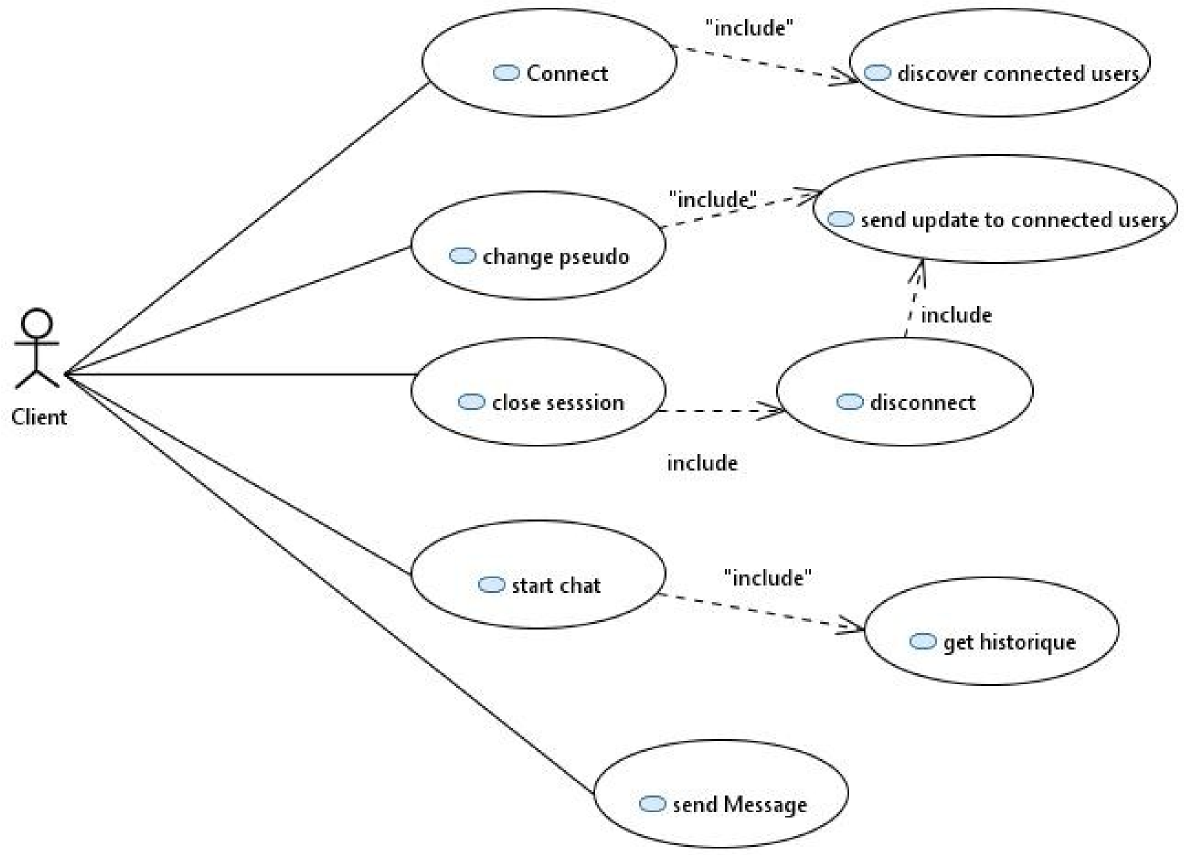


Figure 7 Use Case diagram

2. Class diagrams

The classes derived from our conception and the interaction between those classes are described in the diagram below. It contains the class

a. *Agent*: it represents the user interface between the user and our system. It handles all the actions of the user and calls the corresponding method of the system.

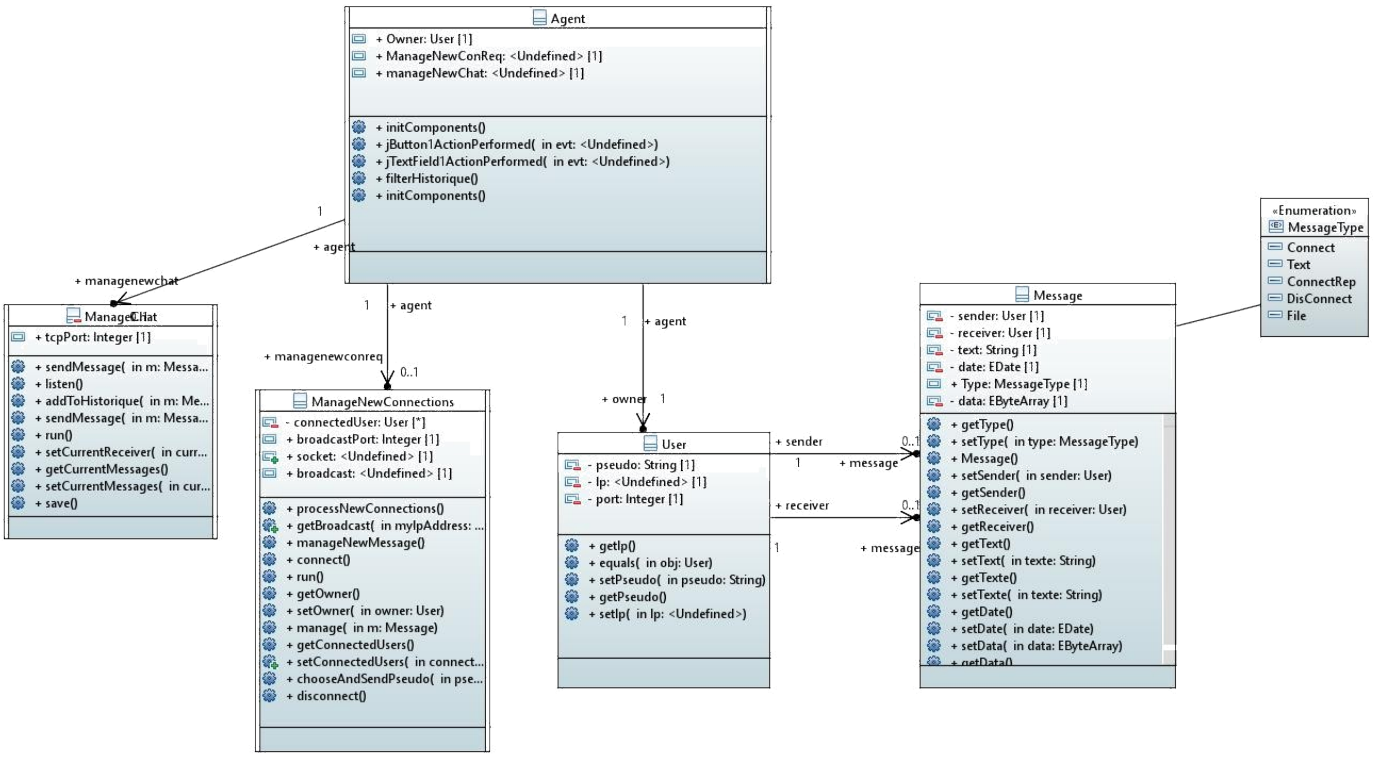
b. *ManageNewConnection*: as its name explains, this class is used to manage new connections. It communicates with other systems to handle new connection requests, new change of pseudo and maintain the current connectedUserList up to date.

c. *ManageChat*: this class is used for the communication between the two systems. It is used to send or receive messages.

d. *User*: this class represents the user in our system.

e. *Message*: it represents a message exchanged in the system.

<u>*MessageType*</u>: it indicates the type of the messages exchanged. A message can be a connection request, a connection response, a file or a text



Agent
+ Owner: User [1]
+ ManageNewConReq: <Undefined> [1]
+ manageNewChat: <Undefined> [1]
+ initComponents()
+ jButton1ActionPerformed(in evt: <Undefined>)
+ jTextField1ActionPerformed(in evt: <Undefined>)
+ filterHistorique()
+ initComponents()
1
+ agent
+ managenewchat
ManageChat
+ tcpPort: Integer [1]
+ sendMessage(in m: Messa...
+ listen()
+ addToHistorique(in m: Me...
+ sendMessage(in m: Messa...
+ run()
+ setCurrentReceiver(in curr...
+ getCurrentMessages()
+ setCurrentMessages(in cur...
+ save()
1
+ agent
+ managenewconreq
0..1
ManageNewConnections
- connectedUser: User [*]
+ broadcastPort: Integer [1]
+ socket: <Undefined> [1]
+ broadcast: <Undefined> [1]
+ processNewConnections()
+ getBroadcast(in myIpAddress: ...
+ manageNewMessage()
+ connect()
+ run()
+ getOwner()
+ setOwner(in owner: User)
+ manage(in m: Message)
+ getConnectedUsers()
+ setConnectedUsers(in connect...
+ chooseAndSendPseudo(in pse...
+ disconnect()
1
+ agent
+ owner
1
User
- pseudo: String [1]
- Ip: <Undefined> [1]
- port: Integer [1]
+ getIp()
+ equals(in obj: User)
+ setPseudo(in pseudo: String)
+ getPseudo()
+ setIp(in Ip: <Undefined>)
+ sender
1
+ message
0..1
+ receiver
1
+ message
0..1
Message
- sender: User [1]
- receiver: User [1]
- text: String [1]
- date: EDate [1]
+ Type: MessageType [1]
- data: EByteArray [1]
+ getType()
+ setType(in type: MessageType)
+ Message()
+ setSender(in sender: User)
+ getSender()
+ setReceiver(in receiver: User)
+ getReceiver()
+ getText()
+ setText(in texte: String)
+ getTexte()
+ setTexte(in texte: String)
+ getDate()
+ setDate(in date: EDate)
+ setData(in data: EByteArray)
+ getData()
«Enumeration»
MessageType
Connect
Text
ConnectRep
DisConnect
File

Figure 8 Class Diagram

3. Sequence diagrams

In the following paragraphs, we will describe our sequence diagrams.

a. Scenario “ Connect ”

This diagram describes the connection scenario. When a user runs the app, a connection request is sent to all connected users. The answer to this request by providing their pseudo. Then the new user can set his pseudo and Agent must check if he can use it or not.

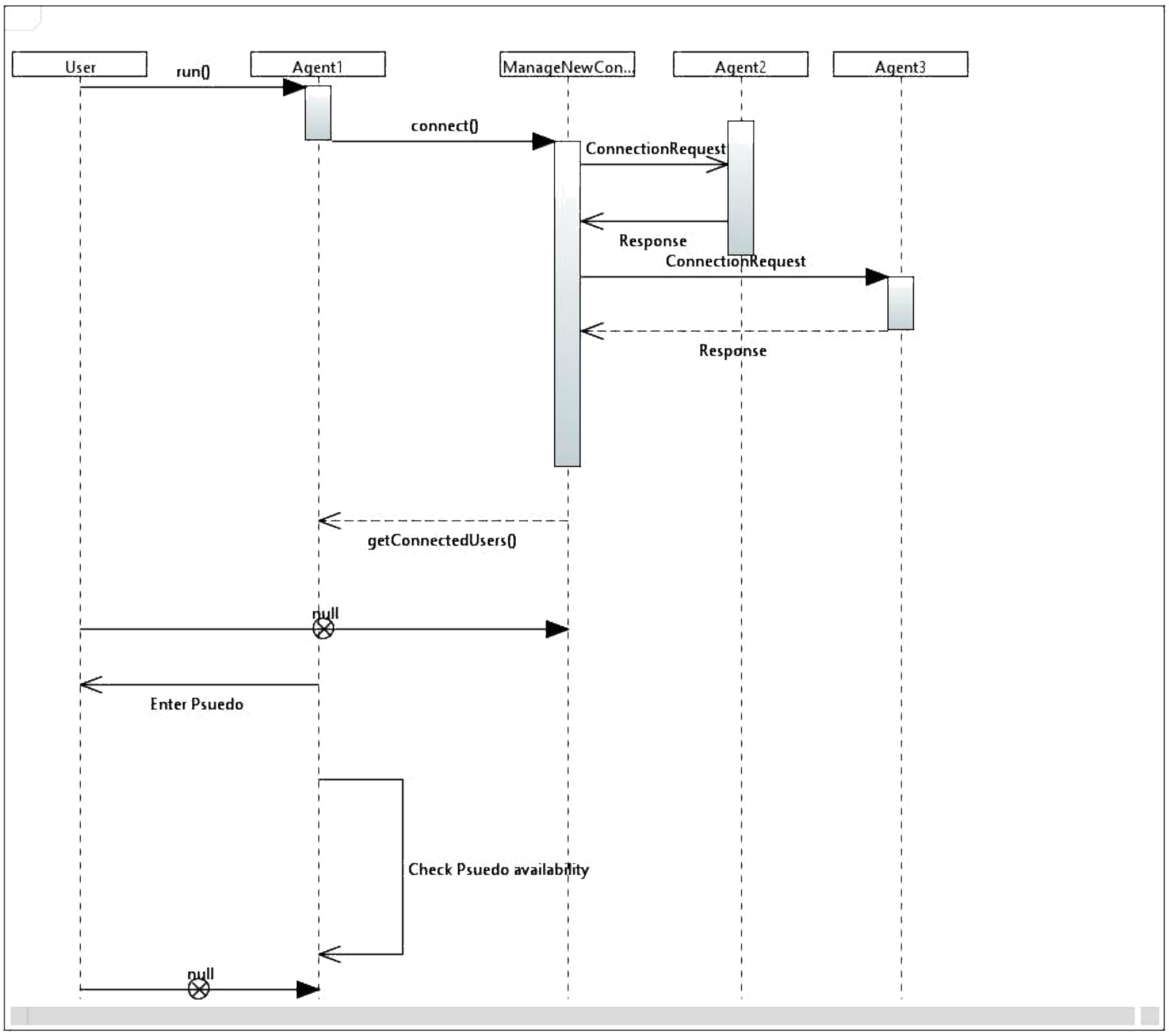


Figure 9 Connection Scenario

Scenario "send message"
In order to discuss with a specific user, a user has to select the name of the host user in connectedUserLIst. After that, the system will display the history of the exchange between those users. Finally, the user can send his messages.

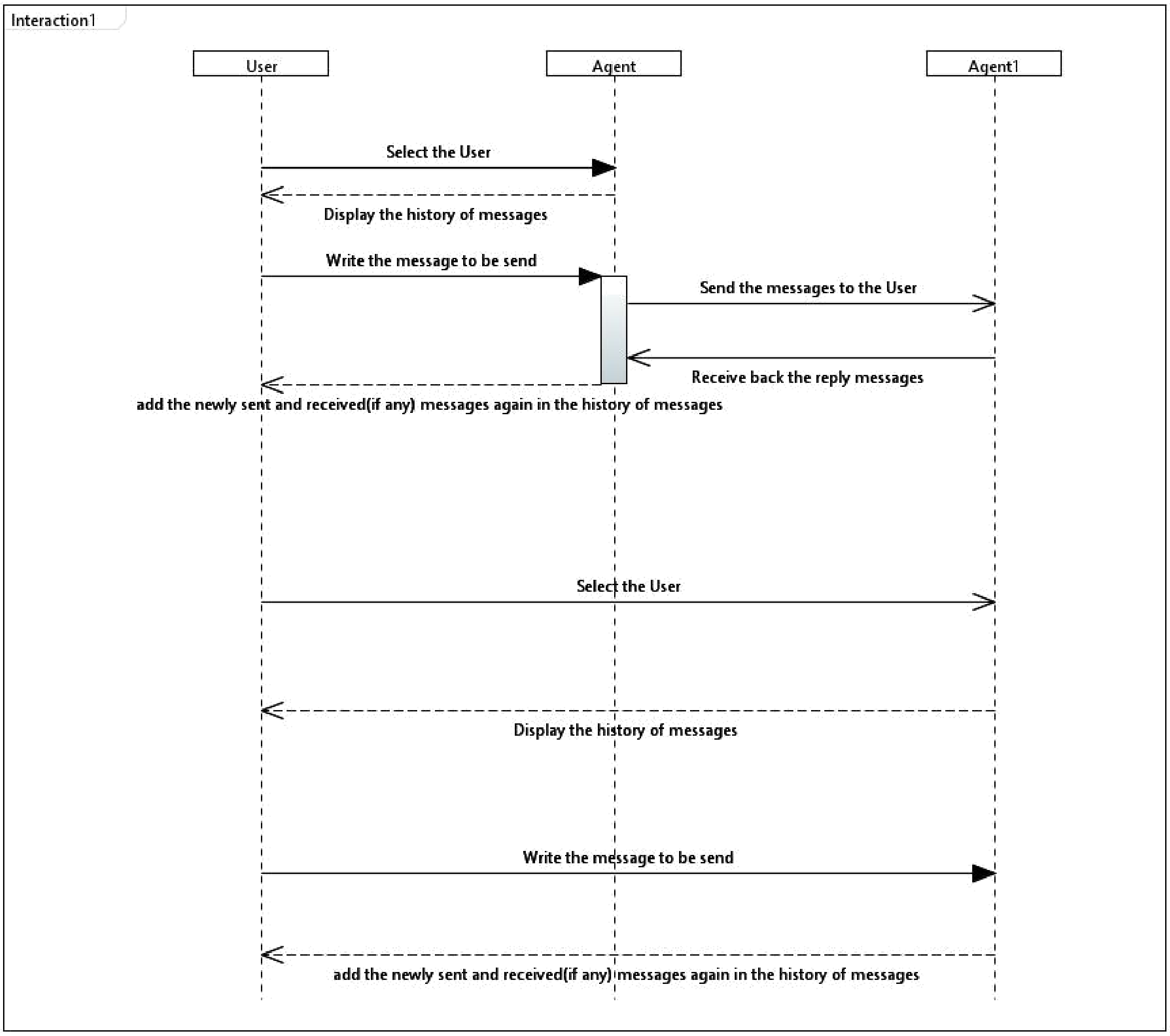

Interaction1
User
Agent
Agent1
Select the User
Display the history of messages
Write the message to be send
Send the messages to the User
Receive back the reply messages
add the newly sent and received(if any) messages again in the history of messages
Select the User
Display the history of messages
Write the message to be send
add the newly sent and received(if any) messages again in the history of messages

Figure 10 Exchange messages with a user

c. Scenario “Change pseudo”

In order to change one’s pseudo, a user will set his new pseudo and the System must check its availability. If the pseudo is available, the user’s pseudo will be updated and the system must send the update to the connected users.

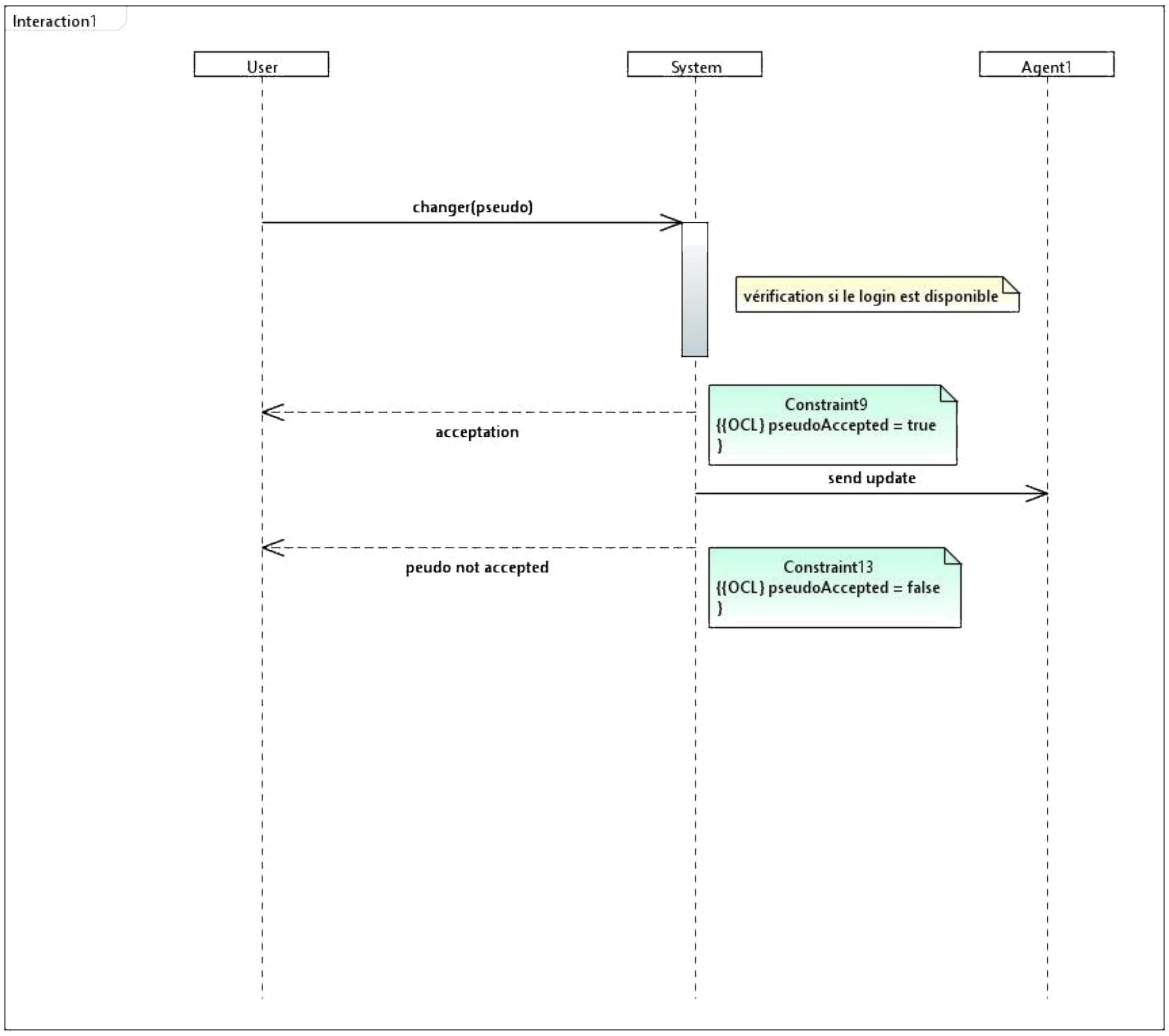


Figure 11: Scenario Change pseudo

d. Close section

When a user closes his section, the system will inform the connected users so they can update their connectedUserList.

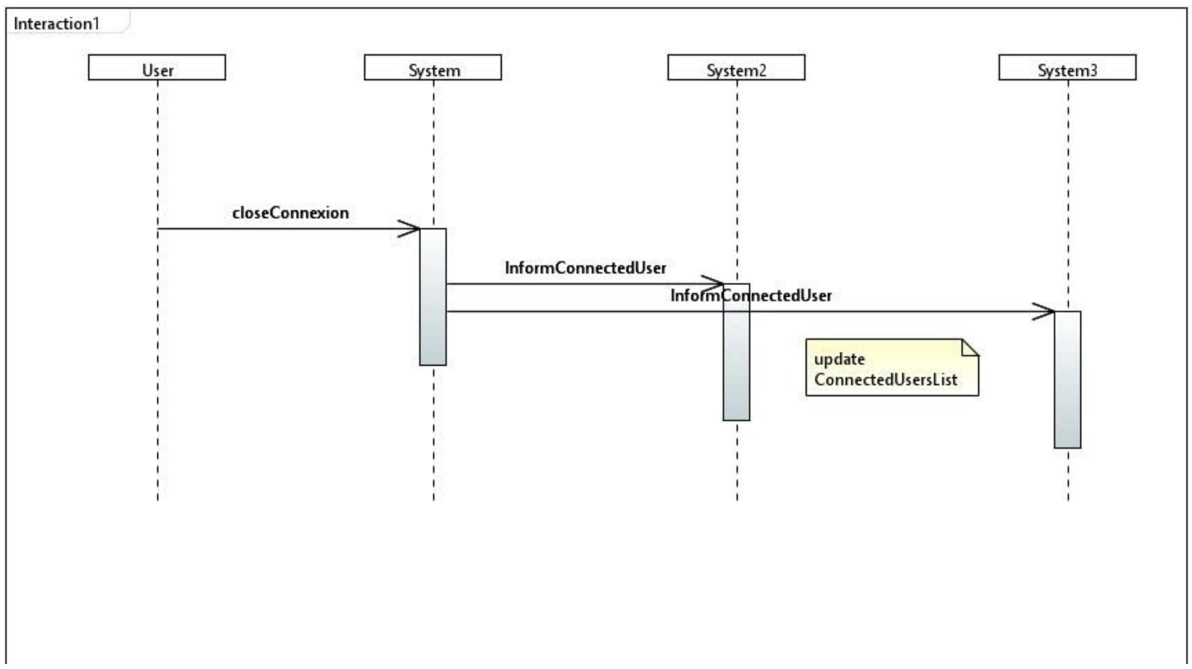


Figure 12 : Scenario Close section

4. Composite structure

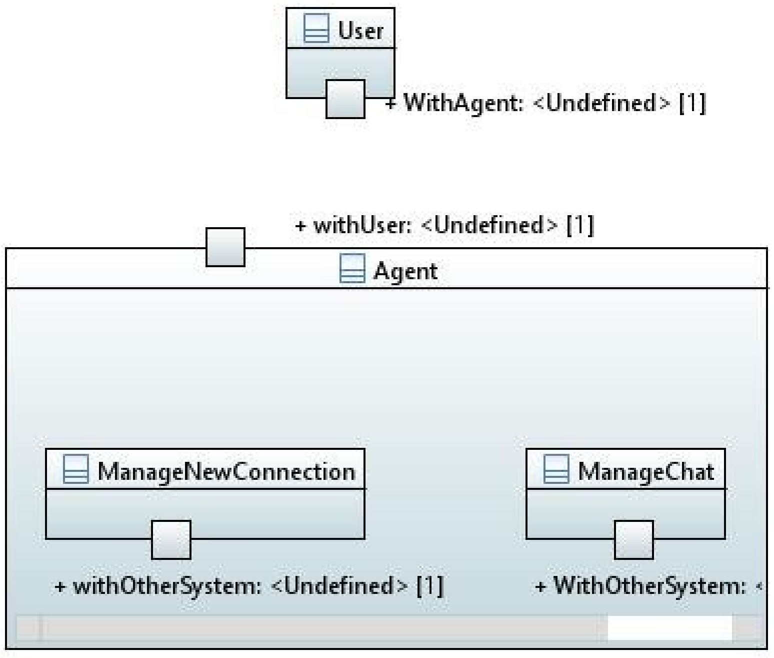

User
+ WithAgent: <Undefined> [1]
+ withUser: <Undefined> [1]
Agent
ManageNewConnection
ManageChat
+ withOtherSystem: <Undefined> [1]
+ WithOtherSystem: <

Figure 13 : Composite Structure Diagrams

5. State machine

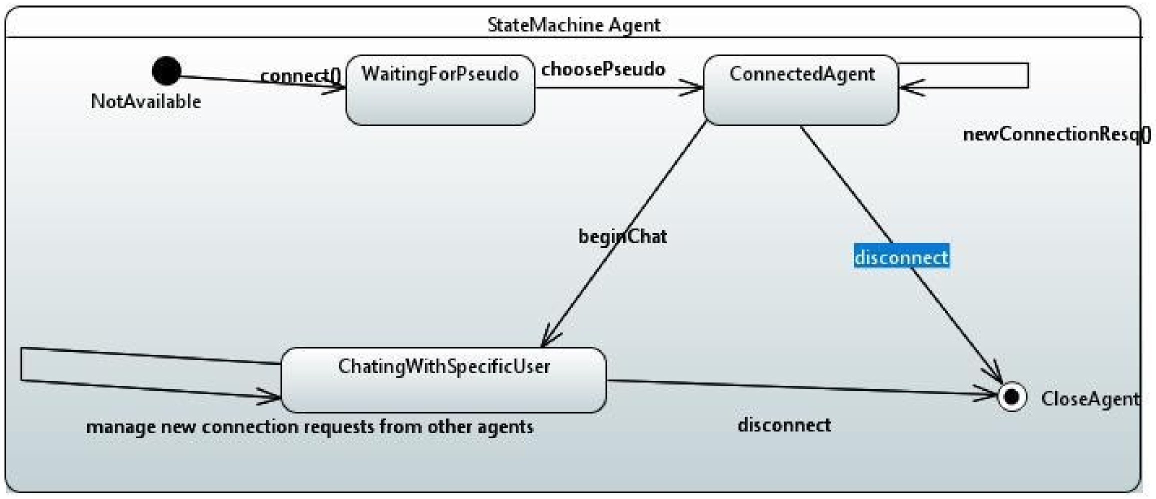


Figure 14 : Statechart Diagrams

VI. Conclusion

This document presents a global view on our chat app realized in java. An app can be used for conversation on a local area network. It has an extension with servlets that allows a remote user to communicate with local users. The document presents how the app works, the tests and conception of the application.