The Report
Of
The Object-oriented
Communication assignment
"The Study Case (Broker& Agents)"
Concurrency Issues

An Object-oriented Communication

2006/2007

By

Salaheddin Darwish

Abstract

Based on the case study requirements, this report aims to discuss the design and implementation of an object-oriented chat system (the case study). Three main entities need to be developed in this system, one broker agent and other two agents of the sender and receiver. The protocols of this system here represent how the communication should be between two given agents (i.e. Users) and a broker (i.e. Server). The roker agent here is responsible for arranging a connection between two agents and then after the session is established, it moves the control o the agents so independently, every two gen exchange messages between each other when they have made a connection between them and to give solutions for the problems of concurrency.

Java is selected to develop this system using particular libraries (particularly java.net and the java.io). Hence, a socket programming in java, one of the tools to build a client-Server application, is used to make establish connections. Finally, I try to evaluate my system how effective and efficient it is.

Contents: **page**

1- Introduction:

The surge of the rapid development in networks (e.g. internet) and programming languages bring about some improvements and come up with tools to solve the problems that experts (e.g. in internet and E-commerce) encounter during applying network and communication systems. One of these improvements and tools is the object-oriented communication as middleware which is using OOP languages(e.g. java) to achieve a type of communication and distribution, and build a protocol between nodes in the networks without going deeply in the low level of networks.

Java is a powerful and versatile object-oriented language that has many libraries like (the java.net and the java.io) and its own trait of portability (i.e. no specific platform, Java needs to be applicable) likes JVM. It provides programmers with some methods and functions that they can comfortably build a distributed application to implement communication for example RMI, Socket, and Thread.

Concurrency issues (i.e. Concurrency occurs when two or more execution flows are able to run simultaneously) are the most critical issue in communicating, and especially in sharing resources. So it can be observed that most of the programming languages like java are struggling to give a solution for the problems of concurrency by using, for instance, thread class, synchronized methods and atomic variables.

Regarding the case study, I am going to use Java to implement case study requirements in a distributed application. This is an example of how to use java tools to make up communication and also solve the problems of concurrency in this case study.

2. Requirements Specification:

The case study describes a scenario of communication between Agents and a Broker in the system and the problems of concurrency encountered, that are as following:

- The system comprises one broker and many agents. The broker should be able to simultaneously deal with many agents that bring about a problem how broker should tackle with all the agents at the same time (I have to find a solution to make the broker act in parallel).
- The broker has to make and save a list for all the agents that are connected to him. This list contains the names of the agents and the status of each agent (available, busy, etc…), broker encounters the problem of reading from and writing on the list at if some agents' statuses are changed concurrently how to deal with this situation. I have to find a solution to manage this shared data.
- Each agent can interact with only one of the other agents at the same time. The agent can select another agent to deal with from the broker list of available agents.
- At the same time, several agents should be able to communicate in pairs independently. The broker has no role and is not an intermediary in this communication between two agents.
- The agent can send messages to another selected agent and can display the messages received from the other agent.
- Each agent can end the connection when a user has decided to end.
- Each agent can run on a different machine. This means that the application is for a network of agents' machines.

After rearranging and classifying the requirements of the case study and make my own assumption, I can specify the primary requirements as below:

1) Application for the **Broker** (i.e. Server Application that will be set up on the broker machine) requires :

- A process to manage the connection between the broker and each agent.
- There is a broker list which is considered as a small database that contains connected agents' details.
- A Connected Agents list includes the IP and the name of each agent and its status (Available, Unavailable).
- The name of each agent is unique in the list because depending on creating an account in any network system, I assume the name is identified each agent.
- An ability to allow many agents to deal and connect to the broker at the same time.
- When a new agent connects to the broker, its IP and name (i.e. unique name) and status should be added to the list.
- When any agent participates in a chat session, its status should be unavailable (busy).

- Providing the available (non-busy) agents list to any agents when it requested. I assume that, because there is no meaning to display all unavailable agents that as an agent, I can not contact them and also I am not able to stay waiting for them until they are released, I have to come later or start a new chat with another available one.
- Monitoring the agent list as shared data to ensure that only one process can change the list (reading/writing) to avoid errors and to achieve data integrity.
- Handling the errors which could take place because of the connection or mistake inserting.

2) Application for the **Agent** (i.e. Client Application that will be set up on each agent machine) which requires:

- User Interface to make the user able to deal with the application. I use the Command line environment (console), it is somehow easy to use.
- Providing the application with the IP of the server and the port to make a connection.
- Providing the broker with a proper gent's name.
- Registering the agent as available for communication.
- Allowing the agent to choose its status as either a listener or a talker.
- The agents can request the broker for a list of all the available agents to choose from one agent to contact.
- The agent can start a communication session with any available agent.
- The agent can stop the communication session at any time.
- The agent can make a chat session with only one other agent.
- When starting a communication session (i.e. the agent-to-agent), there is no task for the broker to do (i.e. not an intermediary in the session).
- The agent can send messages to the other agent during the communication session.
- The agent can display on the screen the messages received from the other agent during the communication session.
- The messages should be displayed on both an agent screen. Each message should be prefixed by the agent name.
- The agent can exit the application and disconnect from the broker.

3. Design Discussion:

In accord with the requirements, the proposed system consists of two main applications. One for the broker works as a server and the other for the agent works as a client. There is a type of protocol that explains how the system should be working by taking into consideration that the system here is a MultiClients -server system and there is shared data (Agents List) that should be monitored to solve concurrency problems and implement data integrity. Moreover, the agent application can make a connection (a chat system "peer to peer") between two agents without intervening with the broker.

Diagrams which is coming later, demonstrate this protocol and how to make this interaction between the server (broker) and each one of the clients (agents) and besides how two gents can interact with each other in a chat system.

3.1 The Broker Process (Flow Chart):

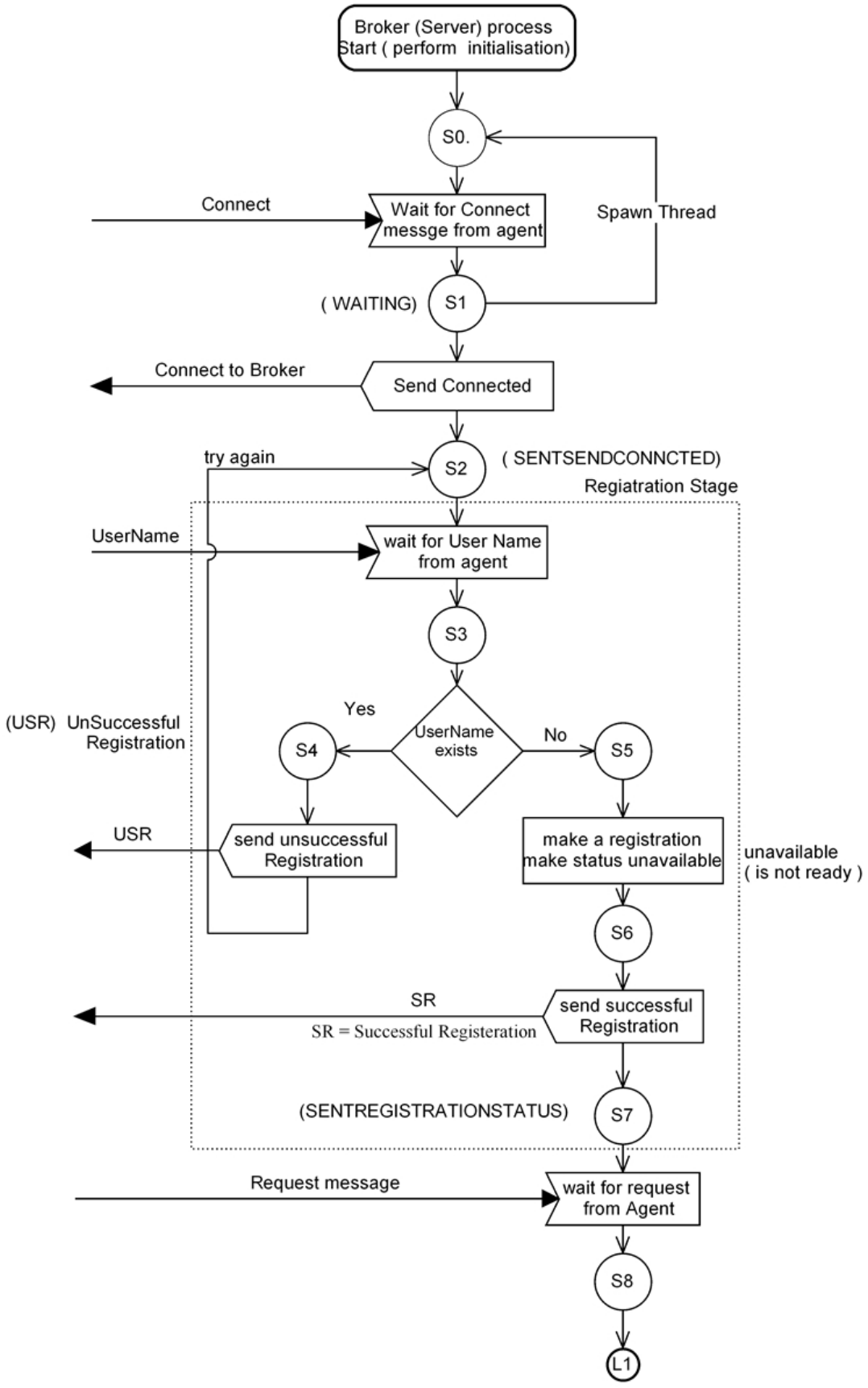
The broker performs the initialisation processes (e.g. creating Server socket) and stays at the state S.0 waiting for a sign of connection from the agent. When the broker receives this sign, it performs a connection between the server and the client by spawning a new thread for this agent, this is to deal with more than one agent at the same time. The broker moves to state S.1 to send "Connect to Broker "to the agent as a sign for accepting a connection. The broker in state S.2 will be waiting until it receives the User Name of the agent to make a registration in its agents' list which includes all connected agents. Then, the broker (the state S3) ensures that this name is sent by the agent is not duplicated in a broker list and if it is duplicated (the state S4) then it will send to the agent ("USR" unsuccessful registration) to inform him to re-enter a new User Name. Otherwise broke creates a new record in the list for this agent and makes its status unavailable (agent is not ready), because its state (talker or listener) have not yet determined to interact and then it sends ("SR" successful registration).

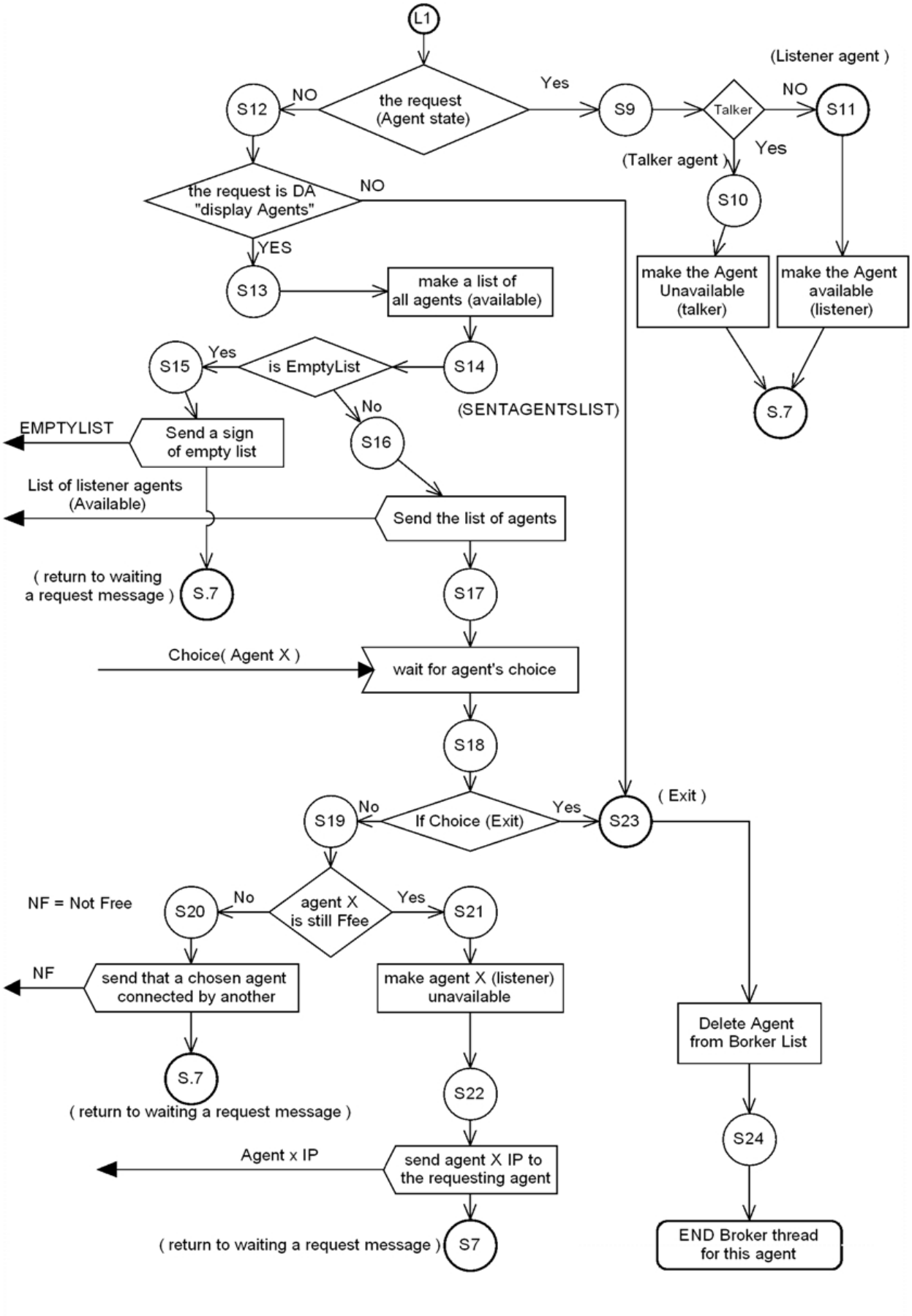
The broker stays in state S.7 waiting for a request message from the agent. If the request message is Talker, which means that the agent wants to connect with another agent, so broker change this agent's status and make him unavailable in the Agents List (S.10) and return to the state S.7 to receive a request message again. If a request message is a listener, the broker will make the agent's state as available for other talker agents in the network and come back to S.7 to wait for the request message.

If the request message is "DA" (i.e. displaying available agents), the broker sends all the available agents (agents who are ready for communication) to the agent. But, after checking its list not being empty. if it is empty then the broker sends a message (EMPTYLIST) after that the broker comes back to the state S.7. If not, The broker stays in state S.15 until receiving a choice or exit. If the choice is the name of the chosen agent "agent X " then the broker checks whether the requested agent is available or not, if it is available, firstly the broker will make him unavailable and secondly, the broker will send IP of this requested agent to the requesting agent(i.e. Talker) and move to the state S.7. Otherwise, the broker sends NF "Not Free"(i.e. unavailable) message to the agent and returns back to state S.7

If the request message is EXIT, the broker deletes the record of this agent from the Agents list and then terminates this agent's thread (the state S.23).

In this design, the state value of the agent is controlled by the agent depending on the its state whether listener or talker, if it is a talker it can himself unavailable and also for another agent who is listener , but listener agent is not .he just can make him self available .
The following diagrams show the protocol in Broker side (his algorithm):


Broker (Server) process Start (perform initialisation)
S0.
Connect
Wait for Connect messge from agent
Spawn Thread
(WAITING)
S1
Connect to Broker
Send Connected
try again
S2
(SENTSENDCONNCTED)
Regiatration Stage
UserName
wait for User Name from agent
S3
Yes
(USR) UnSuccessful Registration
S4
UserName exists
No
S5
USR
send unsuccessful Registration
make a registration make status unavailable
unavailable (is not ready)
S6
SR
send successful Registration
SR = Successful Registeration
(SENTREGISTRATIONSTATUS)
S7
Request message
wait for request from Agent
S8
L1


L1
the request (Agent state)
NO
S12
Yes
S9
Talker
NO
S11
(Listener agent)
Yes
(Talker agent)
S10
the request is DA "display Agents"
NO
YES
S13
make a list of all agents (available)
make the Agent Unavailable (talker)
make the Agent available (listener)
S.7
S14
is EmptyList
Yes
S15
No
S16
(SENTAGENTSLIST)
EMPTYLIST
Send a sign of empty list
List of listener agents (Available)
Send the list of agents
(return to waiting a request message)
S.7
S17
Choice(Agent X)
wait for agent's choice
S18
S19
No
If Choice (Exit)
Yes
S23
(Exit)
NF = Not Free
S20
No
agent X is still Ffee
Yes
S21
NF
send that a chosen agent connected by another
make agent X (listener) unavailable
Delete Agent from Borker List
S.7
(return to waiting a request message)
S22
S24
Agent x IP
send agent X IP to the requesting agent
(return to waiting a request message)
S7
END Broker thread for this agent

3.2. The Agent Processes (flow chart):

When the agent application starts and is provided with the IP address and port of the Broker, it tries to connect to the broker, if the connection is established (i.e. the stateA.0) the agent receives “Connect to broker” message. Therefore, in state A.2 the agent should provide the broker with the User name to open a new record in the broker list of agents and then the agent will wait until it receives an answer whether the registration is successful (“SR”) or unsuccessful (“USR”). if the answer is successful, the agent will move to state A.6 to make the user select the agent state (Talker or listener) or exit.

If the agent chooses the talker state, the agent will then send “Talker” to the broker to make him busy (unavailable) in its recode in broker list and move to the state A.10. In this case, the agent can send the “DA” to the broker to ask for the list of available agents (i.e. free agents who are listeners). Then, the agent is waiting for the list from the broker (the state A11). If it receives a message (EMPTY LISY), it will come back to the state (S6.) to change its state. Otherwise, it has received the list, display the list for the user, user can pick any name from the list (the state A13). Thus, the agent will send the user’s choice to the broker (the state A14). After that, the agent is waiting for IP address of the chosen agent (the state A15).

If the agent receives “NF” (not free), that means the chosen agent becomes unavailable because of being selected by someone, therefore the agent returns to the state (A.6) to give him a chance to change its state.

If the agent receives the IP of a chosen agent by the user, then the agent creates a new thread to start a new chat session with the chosen one, on the other side the chosen one proposed to be listening for a connection. I use that to make a type of autonomy from the connection with the broker and also to keep a link between the agent process and the broker process. If something occurs with the connection of the broker, that will not affect the connection of two agents during the chat session.

In the chat thread, the agent who is a talker will work as a client and the other (i.e. listener) as a server .during the chat session, every two agents can end the chat by saying “end chat “, then automatically the ever thread on both side will be destroyed and return the main agent process for everyone.

If Agent receives “exit” at any receiving point, the agent will terminate the agent process directly (the state A.30) . in this state, the agent will send exit to the broker to let him know that agent’s process is ended.

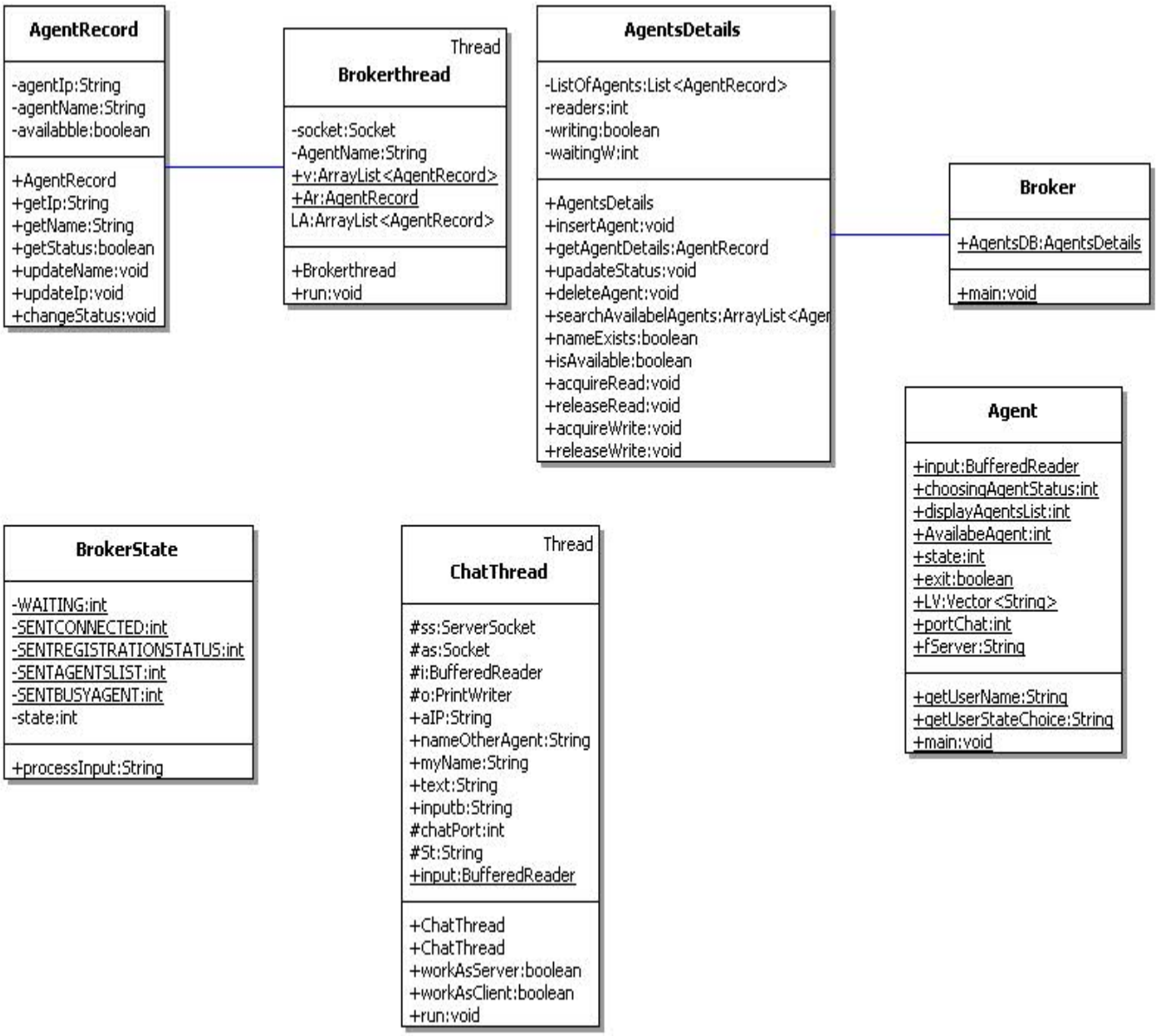
3.3 Class design of the System:

I use this type of design because, I make use of object-oriented language which depends primarily on a class design and moreover, a class design provides the programmer with an excellent approach on how to build an effective and robust software. the programme will be divided in many blocks and deal with this block independently .my system consist of 7 classes :

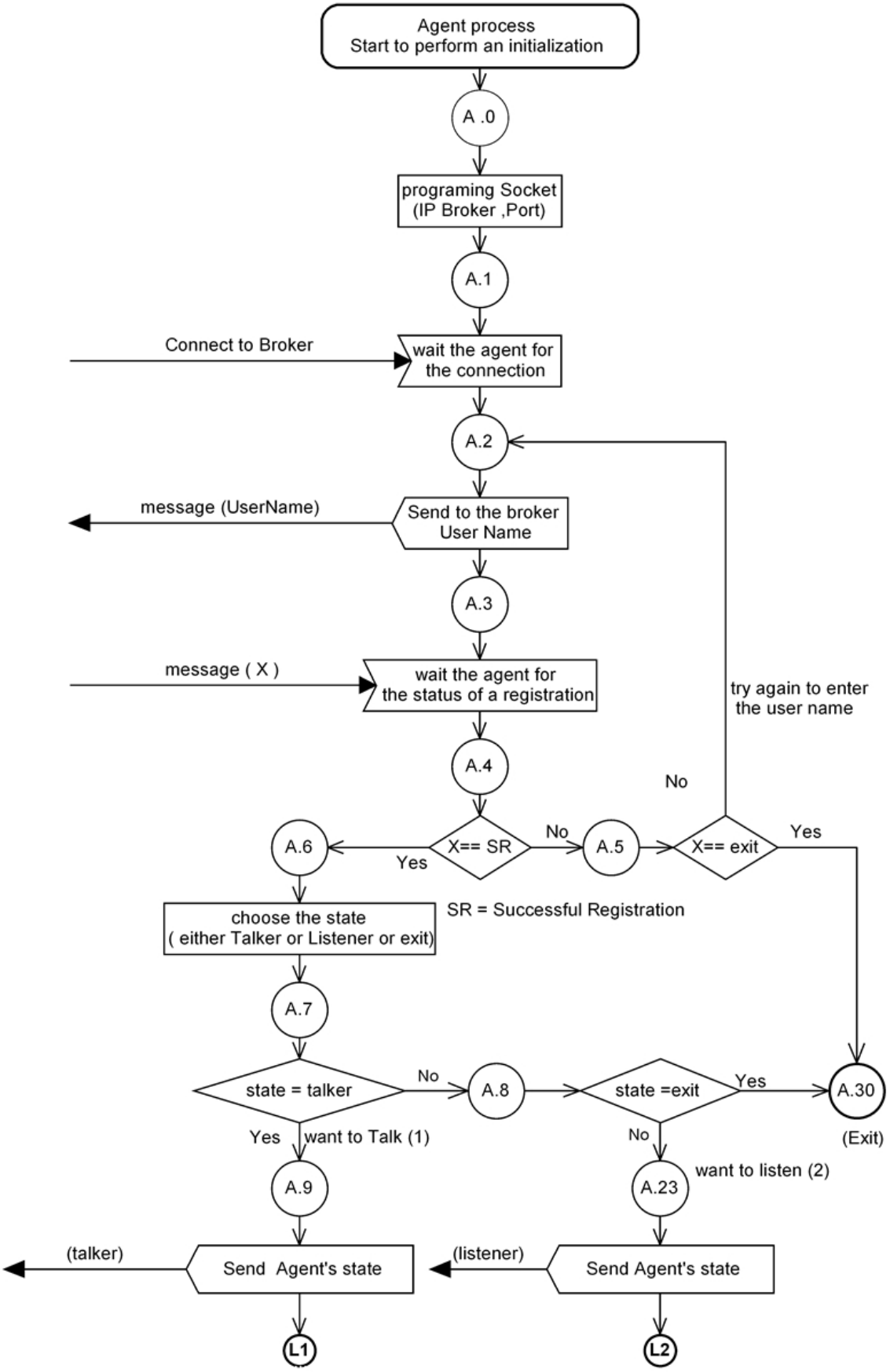
1) Agent class: creates a connection with the broker and other agents
2) Agent Details class: consists of the shared data, methods dealing with it and monitoring method for concurrency issues (e.g. acquireWrite (), releaseRead ()).
3) Agent Record class: form and any type of agent’s information will be dealt with

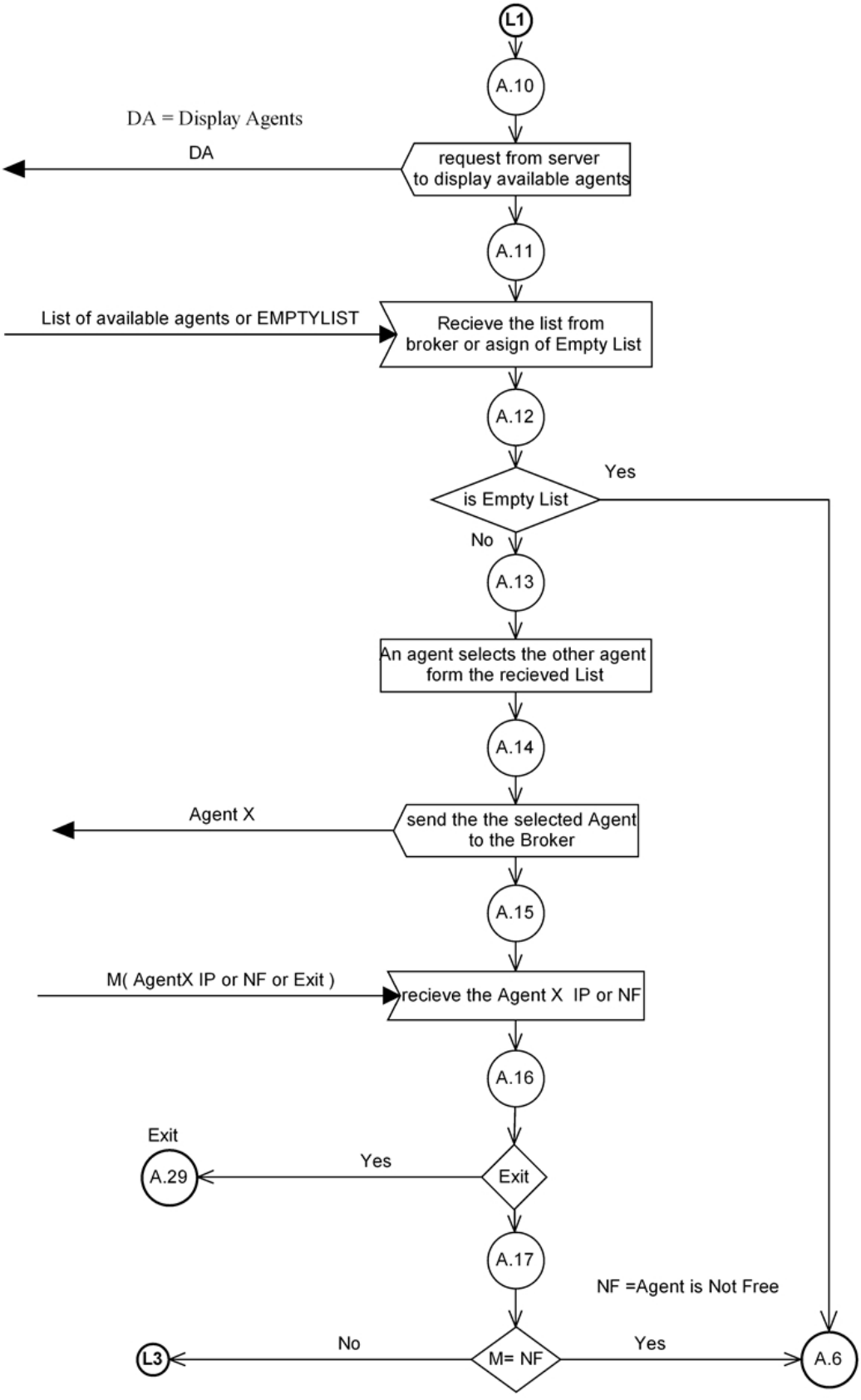
4) Broker class: creates a server process for tackling with the agents.
5) Thread Broker class: create for every agent an own process to implement the form of Multiclients –Server.
6) Broker State: to control the flow of messages that is coming from the agent class
7) Chat Thread: which is responsible to make a chat session with two agents only,

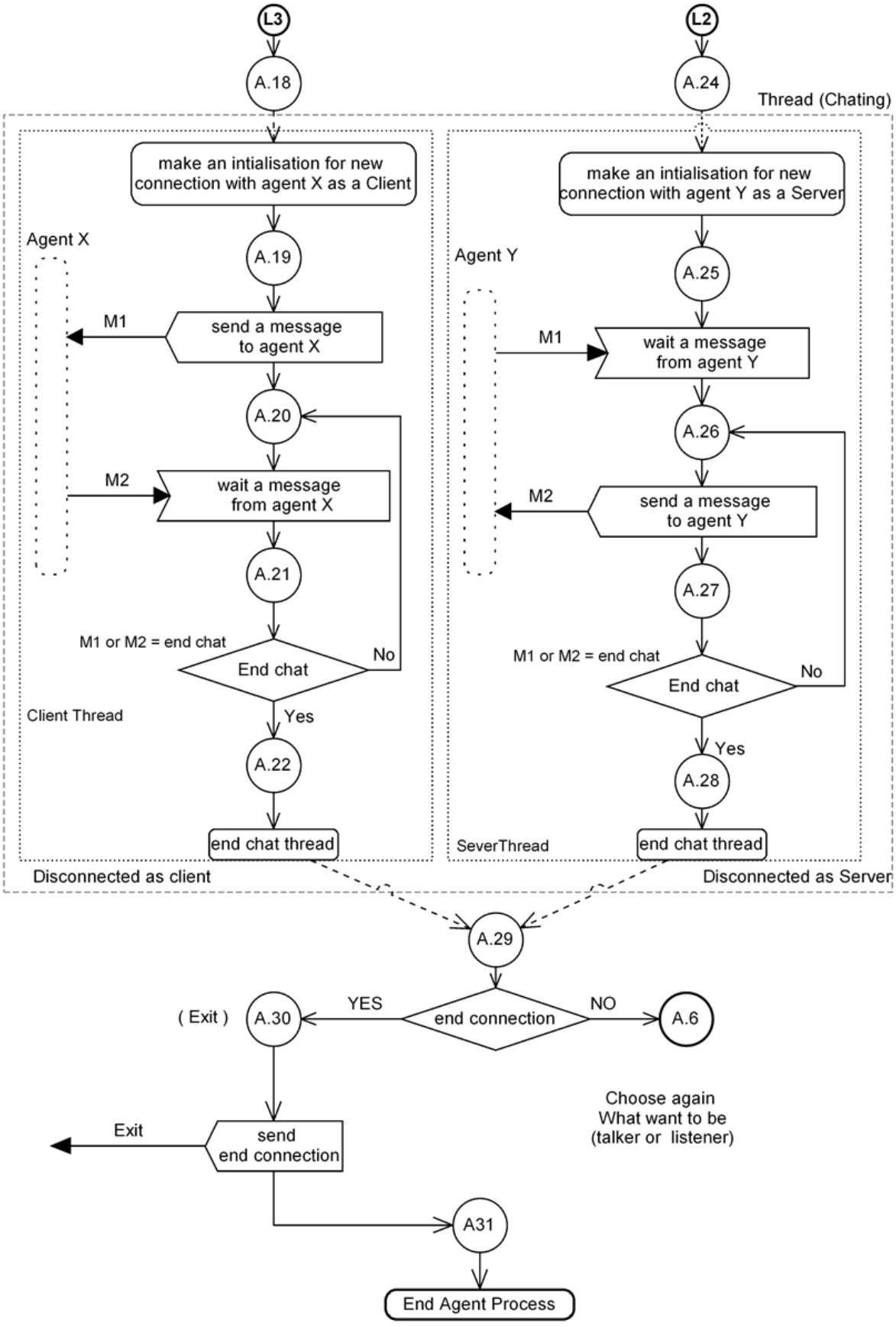
The following diagrams clarify the algorithm:



(Class diagram Multi-Agent-Broker System)


Agent process
Start to perform an initialization
A .0
programing Socket
(IP Broker ,Port)
A.1
Connect to Broker
wait the agent for
the connection
A.2
message (UserName)
Send to the broker
User Name
A.3
message (X)
wait the agent for
the status of a registration
A.4
try again to enter
the user name
No
A.6
X== SR
No
A.5
X== exit
Yes
Yes
choose the state
(either Talker or Listener or exit)
SR = Successful Registration
A.7
state = talker
No
A.8
state =exit
Yes
A.30
(Exit)
Yes
want to Talk (1)
No
want to listen (2)
A.9
A.23
(talker)
Send Agent's state
(listener)
Send Agent's state
L1
L2


L1
A.10
DA = Display Agents
DA
request from server to display available agents
A.11
List of available agents or EMPTYLIST
Recieve the list from broker or asign of Empty List
A.12
Yes
is Empty List
No
A.13
An agent selects the other agent form the recieved List
A.14
Agent X
send the the selected Agent to the Broker
A.15
M(AgentX IP or NF or Exit)
recieve the Agent X IP or NF
A.16
Exit
A.29
Yes
Exit
A.17
NF =Agent is Not Free
L3
No
M= NF
Yes
A.6


L3
A.18
L2
A.24
Thread (Chating)
make an intialisation for new connection with agent X as a Client
make an intialisation for new connection with agent Y as a Server
Agent X
A.19
Agent Y
A.25
M1
send a message to agent X
M1
wait a message from agent Y
A.20
A.26
M2
wait a message from agent X
M2
send a message to agent Y
A.21
A.27
M1 or M2 = end chat
End chat
No
M1 or M2 = end chat
End chat
No
Client Thread
Yes
Yes
A.22
A.28
end chat thread
SeverThread
end chat thread
Disconnected as client
Disconnected as Server
A.29
(Exit)
A.30
YES
end connection
NO
A.6
Choose again
What want to be
(talker or listener)
Exit
send
end connection
A31
End Agent Process

4. Implementation and Deployment:
This program uses the Socket technique to establish the connection between the Broker (server) and each Agent (client). The agent connects to the server through port 4444 and connects to the other agent during a chat through port 4441.

4.1 The broker Classes (a broker side):
Since this program presents multi client-server, The Broker class (Appendix code 1) runs these processes as multi-threading to be enabled to deal with all the clients:

```
while (listening)
    {
      new Brokerthread(serverSocket.accept()).start();
    }
```

Class *Brokerthread* (Appendix A code 2) presents the thread (Brokerthread extends Thread) and control the communication with the agent depending on BrokerState class.

Class *BrokerState* attributes (Appendix A code 3) shows the states that the server could be in depending on the design of the proposed protocol mentioned previously.

AgentsDetails Class (Appendix A code 5) represents the Shared Data in this system and I solve the concurrency problems related to this shared data by using synchronised methods (acquireRead,cquireWrite,releaseRead ,releaseWtite) and monitoring by mutex variables (readers, writing, waitingW).I use the same shared data Model as I used in a lecture. I try to involve this technique in the class methods (procedures and functions) that are used to deal with data for example (you can see all methods in Appendix code 5).

```
public void insertAgent (String ipNewAgent,String newAgenName ) throws
InterruptedException
      {     AgentRecord rA  ;
            try
            {
                  rA = new AgentRecord(ipNewAgent,newAgenName);
                  this.acquireWrite();
                  ListOfAgents.add(rA);
                  this.releaseWrite();
            }
```

I want to mention that I use ArrayList type in java which is a type of collection type to build the shared Data(List of agents in the broker),and it is powerful type because of a dynamic expansion and contraction.*AgentRecord* class is a basic element of Shared data (`ListOfAgents`) .it represents the form of agent 's data (agent IP address , Agent name, available) as a row in a table in the database.
I use in an implementation stage that is not mention in a design stage that how the list of agents' name will be sent, this code below demonstrates that :

```
out.println("ls");
// send a sign for the agent to start send the list

try{sleep(2000);} // delay the process of sending the list
catch (InterruptedException e){ System.err.println(e);}
while (iterator.hasNext())
{
Ar = iterator.next();
```

```
out.println(Ar.getName());}
out.println("le");
```

The broker sends to the agent two signs (list start “ls” and list end ”le”) to make a type of synchronisation between sending by the broker and receiving by the agent to convey correctly the items of the list to the destination.

I attempt to handle most errors in the application by using try /catch instructions. One of the most critical errors on the broker side is when suddenly an agent’s process on the client-side, is terminated, the broker on the server-side should delete the data of this agent.

```
catch (IOException e) { e.printStackTrace();
System.err.println(AgentName +" Thread Shuts Down ....\n "+e);
Broker.AgentsDB.deleteAgent(AgentName); //delete agent }
```

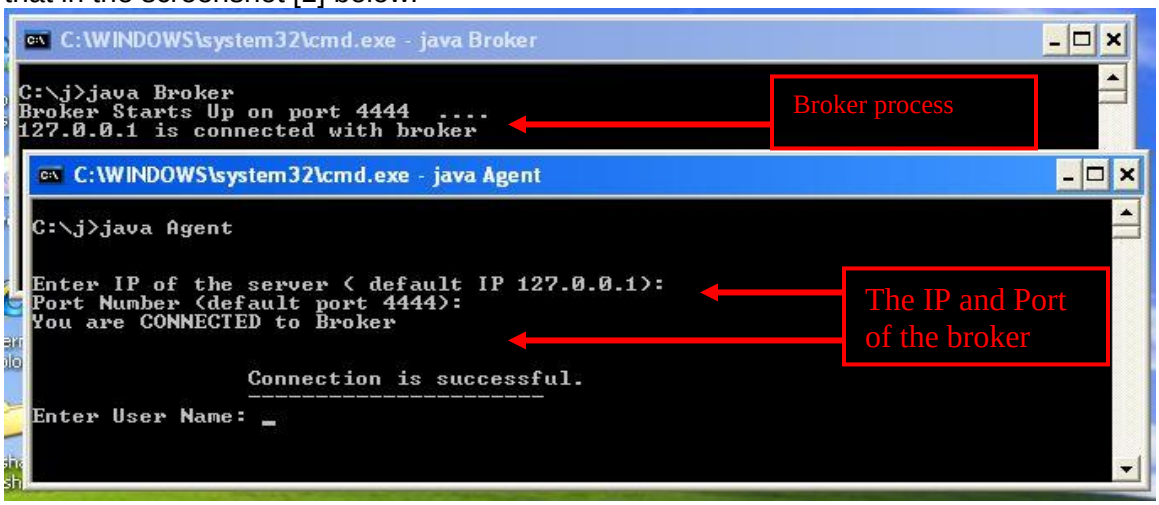
4.2 The agent Classes (an agent side)

There are two classes on the agent side. *Agent* class is responsible for making the connection between the agent and the broker. The *ChatThread* class is responsible for creating an independent session (i.e. thread) between two agents to start chatting with each other .one of them works as a server (listener) and the other client (talker). When the agent during the chat needs to exist, it has to type “chat end” .it will automatically be ended a session and back to the main agent process.

4.3 Testing (Screenshots)

I use the command line environment (console), because I don’t have much experience in GUI in java. Therefore I provide my system with appropriate support to help the users utilise the system’s services. By the screenshots, I try to demonstrate some of the functions and states faced in the program (Note: the rest screenshots in appendix A).

- You can shut down the system by using Ctrl+C.
- First, you have to start up the broke to make the system work well The broker is waiting on port 4444 for any connection. the broker prints out all the events and agents’ messages on the screen. when the agent application is started up, it asks you to enter the IP and port of the broker .there are default values, you can use them. you can see that in the screenshot [1] below:



Screenshot [1]

- At this stage, I am going to explain almost the functions and the states of each application (broker and agent)

- You have to enter the user name to make a registration in the broker.
- If a user name that you have entered is not duplicated in a list of agents, the broker will reply to you with successful registration (i.e. saving your user name) .or reply with unsuccessful registration because of replication (see screenshot[2])
- In this stage after registration, you have to choose your state (listener or talker) if you choose listener by press 2 then automatically you will create a chat thread and you will be in waiting for someone to talk with you (see screenshot [3], code 6).
- If you choose a talker state by press 1, then you have to request the agents' list in the broker by press 3 .and after that you will see available agents to connect with if there are no agents the application informs you that and give you a chance to change your state or exit (see screenshot [4]). If the list is not empty, at this moment you should pick one of them by typing its name and press enter. The system already checks your input to make sure that you enter the name correctly .after all that you automatically will start a chat thread to talk with one, you have chosen. (see screenshot [5], code 6).

- During the chat session. if each of two agents types "chat end", the application will automatically end the session and come back to the main agent process and the system asks you if you need to end your application. If yes, then, the agent process will be terminated and at the same time, the "exit" will be sent to the broker to inform about ending the connection with this agent (see screenshot [5]).

- At any point, if you want to exit, just type exit or 0 depending on the support text in the application. Except for the chat session, you have to type end chat to end it .

5. Evaluation.

5.1 Issues and limitation :

After testing the programme, I believe that all the requirements in the case study are achieved. The application can run on the same or different machine. However, I admit that my application is not a perfect one. This is because I have used a command-line environment (console) which has some limitations. Firstly, this environment relies on typing the instruction that makes somewhat users confused. Secondly, it depends on a sequential control by typing. therefore I have encountered some difficulties in some cases to change the control or to check what the agent inputs. I almost figure out them but there are a few of them not being solved for example, in the agent application when the agent becomes a listener and moves to a waiting state. I could not find the right way to stop this waiting (accept ()) in this environment. Furthermore, when a chat session gets started, the chat between the agents is sequential. The agent can send once and can't send again until it receives a message from the other agent.

Another issue ,the monitoring shared data is somehow working well but in some cases the broker faces problems in waiting (readers) all the writer one that will make some type of impediment for other processes .

I have used in broker application a list array type which is not permanent storage and its size depends on the memory size .there is a risk that if something occurs with broker process make him shut down, all the data of gents will be erased.

5.2 Improvements :

To avoid a serial input for instruction, we have to use the Graphical User Interface which enables the User to handle the program easily and provide the programmers with some type of flexibility in programming its application.

For awaiting issues for the listener agent in the chat session, at the last moment I find a partial solution, by providing the accept () with a timeout that when the time is finished the process interrupts this waiting and return to the main process .

For temporary storage problems, we can use either files or a database to permanently save the data. There are a lot of database systems that can deal superbly with shared data.

6. Conclusion:

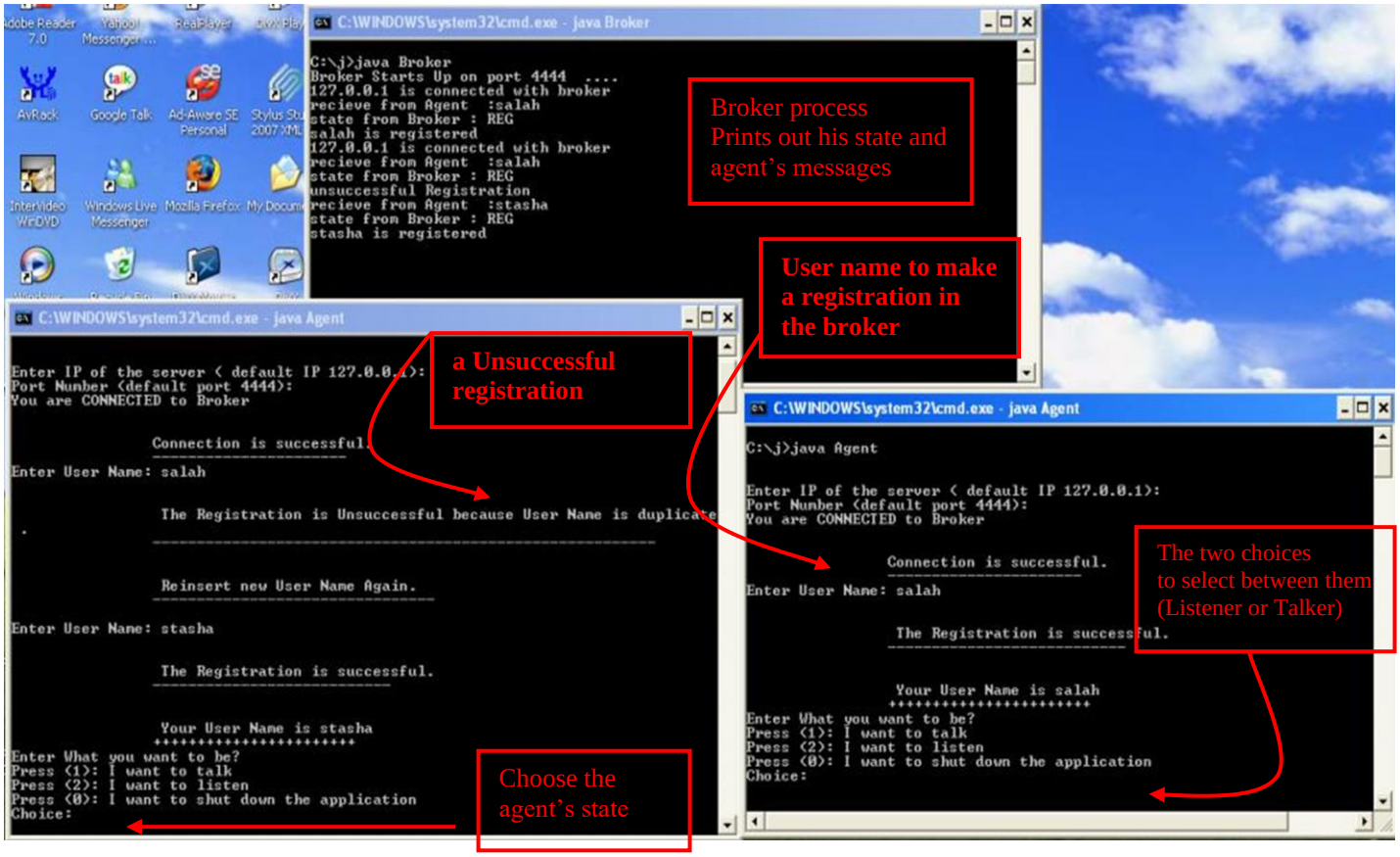
Object-Oriented programming receives an excellent reputation. No one can deny its role of providing the experts with perfect solutions for complicated problems, especially in communication systems. I found that java as the powerful object-oriented programming language is improving every day and is still considered the top of the programming languages because it comes up with some techniques to solve the problems of concurrency.

I enjoyed developing this programme although I don't have much experience in developing network applications and I have firstly encountered some difficulties that I have felt that I could not make it. Finally, I get familiar with java and those concepts about networks and concurrency. In my opinion, I have done very good work.

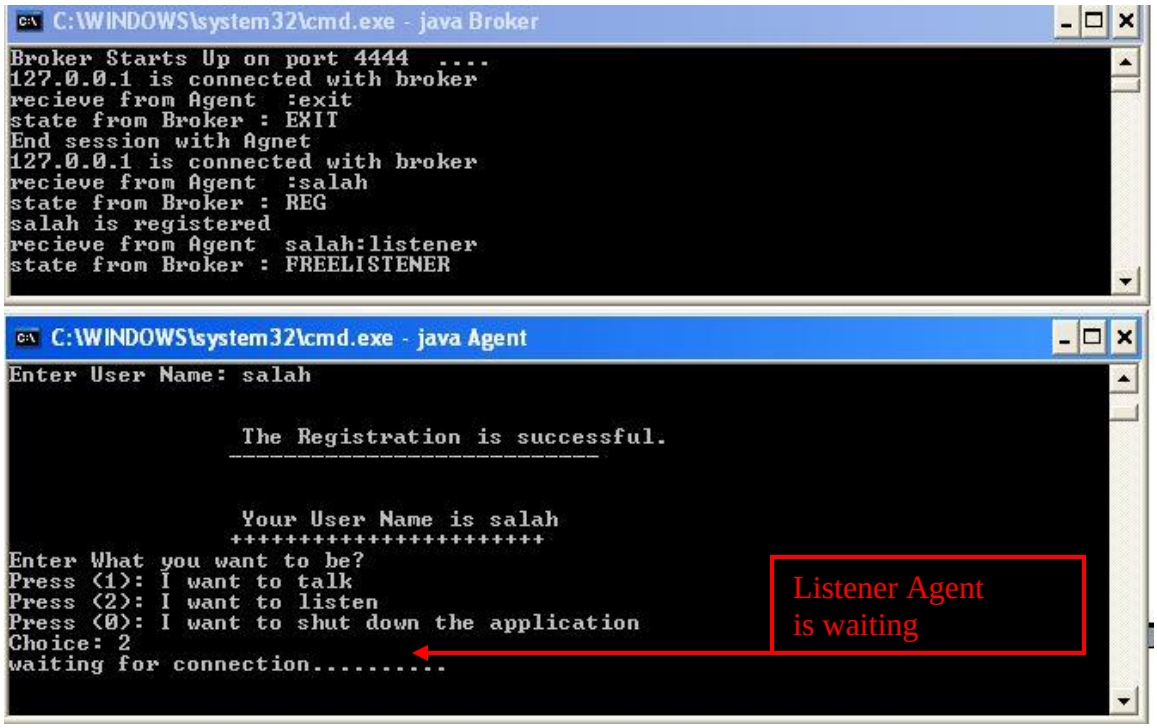
Email: salaheddin.darwish@brunel.ac.uk

Appendix A :

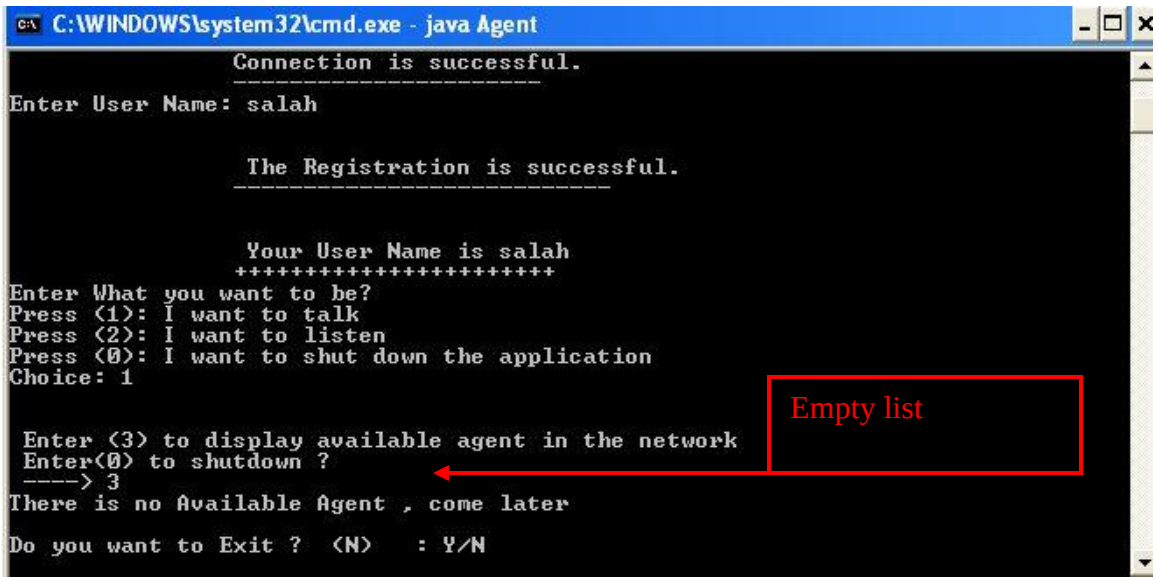
🕮 Screenshots :



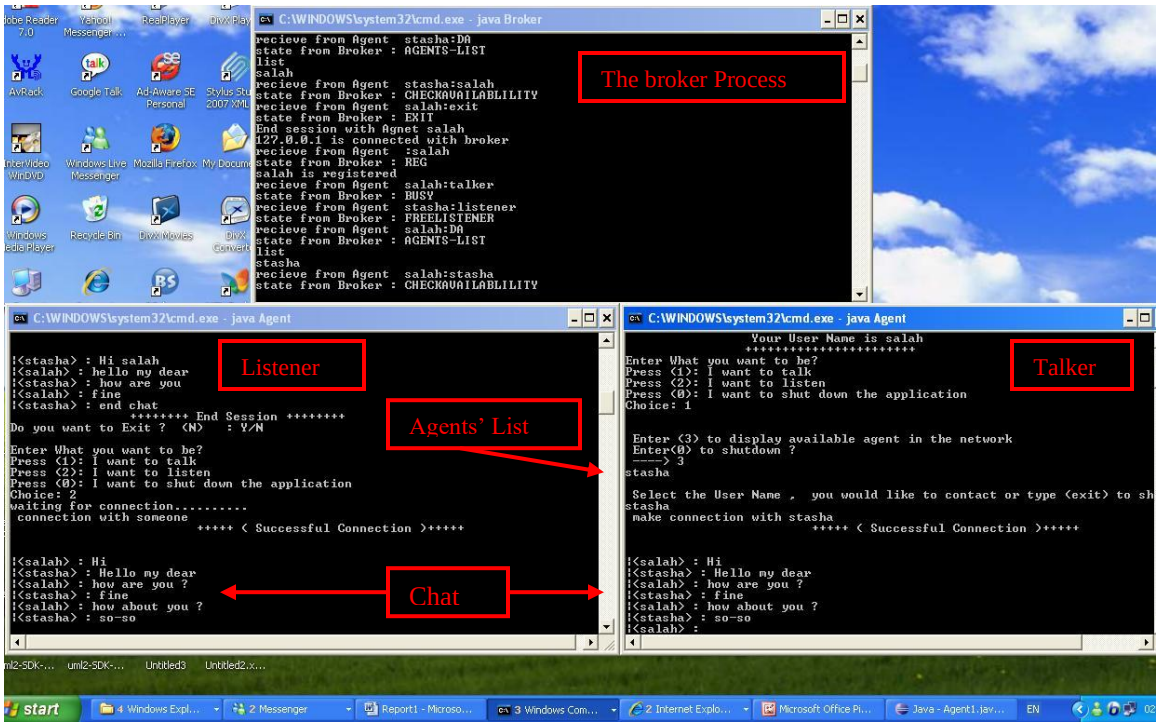
Screenshot [2]



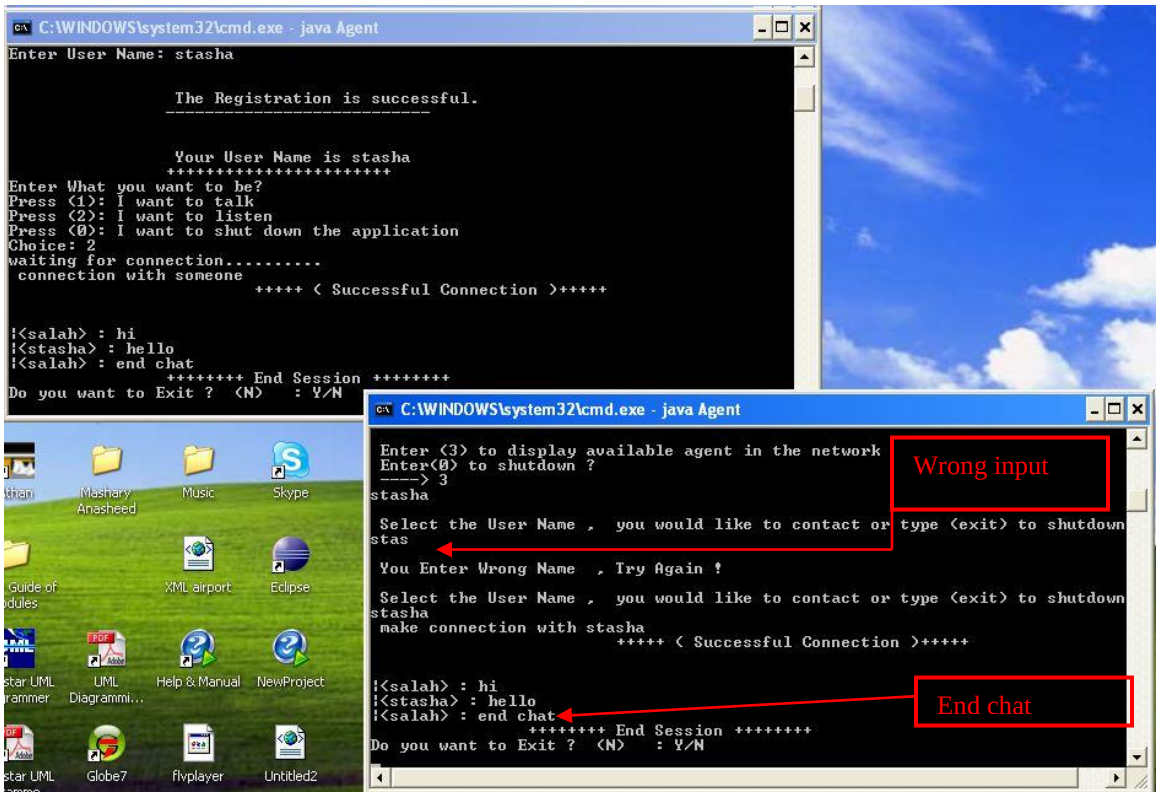
Screenshot [3]



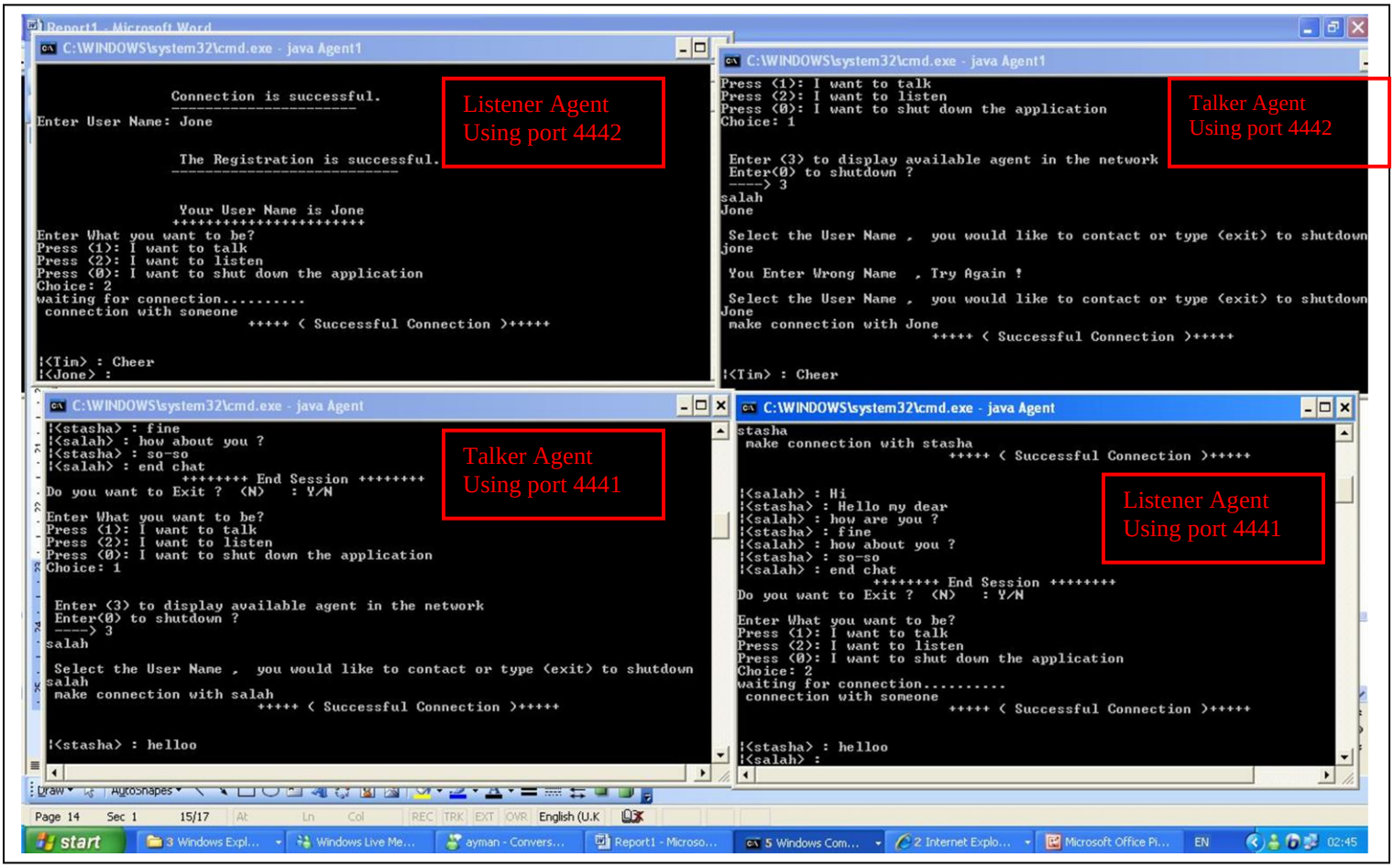
Screenshot [4]



screenshot [5]



screenshot [6]



Screenshot (4 Agents communicating) [7]

🕮 The Source Code

Broker Class (code1):

```
/*
Class kkmsserver taken from KKMultiServer online in
Campione, M. and Walthrath, K (1998) The Java Tutorial
http://java.sun.com/docs/books/tutorial/networking/sock
ets/example-1dot1/KKMultiServer.java
Modified by Simon Taylor (1998) for CS3024A NODS
I modified to implement my assignment (salaheddin Darwish )
*/

import java.net.*;
import java.io.*;
public class Broker {
public static AgentsDetails AgentsDB = new AgentsDetails ();
public static void main(String[] args) throws
IOException
{
//Set up the local variables
ServerSocket serverSocket = null;
boolean listening = true;
// Make the server socket
try {
      serverSocket = new ServerSocket(4444);
      } catch (IOException e) {
System.err.println("Could not listen on port:4444.");
System.exit(-1);
}

/*Whenever a client attempts to connect on the serversocket setup
above, setup a new connection.
Do this forever The new connections are made by starting a new
execution thread Brokerthread.

The argument serverSocket.accept means that the thread will
connect to whichever client demanded the connection The start command
just tells the kkmsthread object
to begin execution.Every thread object has a run() method instead of a
main() method. Start() just tells the object to use
the run() method to begin execution. These threads will be timesliced
by the operating system
*/
System.out.println("Broker Starts Up on port 4444  .... ");
while (listening){
try {
      new Brokerthread(serverSocket.accept()).start(); // S.1
      } catch (SocketException e )
      {    System.err.println("Server Shuts Down ....");
            System.exit(1);
      }

}
serverSocket.close();
}
}
```

Broker Thread Class (code 2) :

```
/*
Class kkmsthread taken from KKMultiServerThread online
in Campione, M. and Walthrath, K (1998) The Java
Tutorial http://java.sun.com/docs/books/tutorial
/networking/sockets/
example-1dot1/KKMultiServerThread.java
Modified by Simon Taylor (1998) for CS3024A NODS
I modified by salaheddin  Drwish for the assignment of object oriented
communication

*/

import java.net.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.io.*;
public class Brokerthread extends Thread {
// Variable declarations
private Socket socket = null;
private String AgentName ="";
public static ArrayList<AgentRecord> v;
public static  AgentRecord Ar = new AgentRecord(null, null)  ;
ArrayList<AgentRecord> LA = new ArrayList<AgentRecord> ();
//Constructor method
public Brokerthread(Socket socket)
{
super("Brokerthread");
this.socket = socket;
}
public void run()
{
try
{    // sender side
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
// reciever Side
BufferedReader in = new BufferedReader( new
InputStreamReader(socket.getInputStream()));

String inputLine ="",
outputLine ="" ,
ip         = "";
;


BrokerState kks = new BrokerState();
outputLine = kks.processInput(null);
out.println(outputLine);
ip =socket.getInetAddress().getHostAddress().toString();
//System.out.println(inputLine);
System.out.println(ip+" is connected with broker" );
l:    while ((inputLine = in.readLine()) != null)
{
```

```
System.out.println("recieve from Agent  "+AgentName+":"+ inputLine);

outputLine = kks.processInput(inputLine);

System.out.println("state from Broker : " + outputLine);

if (outputLine.equals("REG"))
{
if (! Broker.AgentsDB.nameExists(inputLine))
{
try
{
Broker.AgentsDB.insertAgent(ip ,inputLine );
System.out.println(inputLine+" is registered");
AgentName = inputLine;

} catch (InterruptedException e)
{ System.err.println(e);}
out.println("SR");
outputLine = kks.processInput("SR");

}
else
{
out.println("USR");
System.out.println("unsuccessful Registration");
outputLine = kks.processInput("USR");
}
}
else
{
if (outputLine.equals("BUSY")) // talker
{

Broker.AgentsDB.upadateStatus(AgentName , false); // make him busy
continue l ; // continue the While Loop
}
else if (outputLine.equals("FREELISTENER"))  //make him free listener
who need t interact

{
Broker.AgentsDB.upadateStatus(AgentName , true); // make him free
continue l ; // continue the While Loop
}
else if (outputLine.equals("AGENTS-LIST"))

{
v = Broker.AgentsDB.searchAvailabelAgents() ;
Iterator<AgentRecord> iterator = v.iterator();
if ( v.isEmpty() )
{
outputLine = kks.processInput("EMPTYLIST");
out.println("EMPTYLIST") ;
// System.out.println("EMPTYLIST");
}
else{
out.println("ls"); // send a sign for the agent to start send the list
```

```
// delay the process of sending the list

try{sleep(2000);}
catch (InterruptedException e){ System.err.println(e);}

while (iterator.hasNext())
{
System.out.println("list");
Ar = iterator.next();
System.out.println(Ar.getName());
out.println(Ar.getName());
}
out.println("le");
continue l ;
}
}
else if (outputLine.equals("CHECKAVAILABLILITY"))
{
if ( Broker.AgentsDB.isAvailable(inputLine))
{
try{
Ar= Broker.AgentsDB.getAgentDetails(inputLine);
Broker.AgentsDB.upadateStatus(inputLine, false); // the state of chosen
agent
} catch (InterruptedException e){ System.err.println(e);}

out.println(Ar.getIp());//send Ip to the agent
outputLine = kks.processInput("as");
} else
{
out.println("NF"); //send Not Free for a chosen Agent to the agent

outputLine = kks.processInput("uas");
}
}
else if (outputLine.equals("EXIT"))

{
out.println("exit");
Broker.AgentsDB.deleteAgent(AgentName);
//delete information of this agent because of exit
break;

}
}


} // end loop

System.out.println("End session with Agnet "+AgentName);
out.close();
in.close();
socket.close();
}
catch (SocketException e )
{
System.err.println(AgentName +" Thread Shuts Down ....");
```

```
Broker.AgentsDB.deleteAgent(AgentName); //delete agent
}
catch (IOException e) { //System.err.println("Thread Shuts Down ....");
e.printStackTrace();
System.err.println(AgentName +" Thread Shuts Down ....\n "+e);
Broker.AgentsDB.deleteAgent(AgentName); //delete agent
}
}
}
```

Broker State Class (code 3):

```
/* The class is made to give any status of the broker " any step that
protocol have reached "*/
public class BrokerState


{
private static final int WAITING = 0;//S.1
private static final int SENTCONNECTED = 1;//S.2
private static final int SENTREGISTRATIONSTATUS = 2;//S.6
private static final int SENTAGENTSLIST = 3;//S.9
private int state  = WAITING;


public String processInput(String theInput)
{
String theOutput = null;
switch(state)
{
case WAITING://S.1
theOutput = "CONNECTED to Broker";
state = SENTCONNECTED;//Change state to S.2
break;

case SENTCONNECTED://S.2
//Register this agent as busy
//Change state to S.22
if(theInput.toLowerCase().equals("exit"))
{
theOutput = "EXIT";
state = WAITING ;//Change state to S.1
}
else if (theInput.equals("USR") )
{
state = SENTCONNECTED ;//Change state to S.2

}
else if (theInput.equals("SR"))
{
state = SENTREGISTRATIONSTATUS ; //Change state to S.7

}
else
{
theOutput = "REG"; // first step of registration S.3
```

```
}
break;

case SENTREGISTRATIONSTATUS ://S.7

if(theInput.toLowerCase().equals("talker"))
{
theOutput = "BUSY";
//Register this agent as busy
state = SENTREGISTRATIONSTATUS ;//Change state to S.7
}
else if(theInput.equals("DA"))
{
theOutput = "AGENTS-LIST";
state = SENTAGENTSLIST;//Change state to S.14
}
else if (theInput.toLowerCase().equals("listener"))
{
theOutput = "FREELISTENER";
state =SENTREGISTRATIONSTATUS ;//Change state to S.21
}
else if (theInput.toLowerCase().equals("exit")){
theOutput = "EXIT"; //Change state to S.23
state = WAITING;
}
break;


case SENTAGENTSLIST://State S.14
if(theInput.toLowerCase().equals("exit"))
{
theOutput = "EXIT";
state = WAITING;//Change state to S.1
}
else if( theInput.toLowerCase().equals("uas"))
{
theOutput = "NF";
state = SENTREGISTRATIONSTATUS ; //Change state to S.7
}
else if (theInput.toLowerCase().equals("as"))
{
theOutput = " Sent sIP";
state = SENTREGISTRATIONSTATUS ; //Change state to S.7
}
else if (theInput.equals("EMPTYLIST")) // Empty List  its mind S.15
{
theOutput = "EMPTYLIST";
state = SENTREGISTRATIONSTATUS ; //Change state to S.7
}
else
{
theOutput = "CHECKAVAILABLILITY";
state = SENTAGENTSLIST ; //Change state to S.14
}

break;
```

```
return theOutput;

}
}
```

Agent Record Class (code 4):

```
/* this class makes the form of a record for the agent " Agent's
information
 * it is used basically in creating the shared list of agents connected
to the broker
 */
public class AgentRecord {

	private  String agentIp  ;
	private  String agentName  ;
	private  boolean availabble   ;

	public AgentRecord (String agentIp , String agentName )
	{
		this.agentIp=agentIp ;
		this.agentName=agentName ;
		this.availabble= false ;

	}// end constructor

	public String getIp ()
	{
		return (this.agentIp);
	}

	public String getName ()
	{
		return (this.agentName);
	}

	public boolean getStatus ()
	{
		return (this.availabble);
	}
	public  void updateName(String N )
	{
		this.agentName = N ;

	}
	public  void updateIp(String N )
	{
		this.agentIp = N ;

	}
	public  void changeStatus(boolean B)
	{
	  this.availabble= B;

	}
```

```
      }// end class
```

Agent Details Class "Shared Data" (code 5):

```
/**
 * @author cspgssd salaheddin Darwish
 * assignment Object-oriented Communication
 * I have used Simon JE Taylor  Code (2001) in Tutorial #6 about
managing Shared Data * Modified By Salaheddin Darwish
 */
import java.util.*;
public class AgentsDetails {


      private   List<AgentRecord> ListOfAgents ; // the Shared Data
      private int readers=0; // number of concurrent  readers
      private boolean writing=false; // true a thread is writing, false
otherwise
      private int waitingW=0; // number of waiting writers
      // Constructor
      public  AgentsDetails ()
      {
            // intialize the Shared Data
            ListOfAgents = new ArrayList<AgentRecord> ();

      }


      public void insertAgent (String ipNewAgent,String newAgenName )
throws InterruptedException
      {
            AgentRecord rA  ;
      // This method to insert Information about new Agent connected to
the broker

            try
            {
                  rA = new AgentRecord(ipNewAgent,newAgenName);
                  this.acquireWrite();
                  ListOfAgents.add(rA);
                  this.releaseWrite();
            }
            catch(Exception e)
            {     System.err.println(e);        }

      }// end inserAgent

      public AgentRecord getAgentDetails (String naAG)throws
InterruptedException
{
      AgentRecord rA = null  ;
      // This method to insert Information about new Agent connected to
the broker
      Iterator<AgentRecord> iterator = ListOfAgents.iterator();
      try
      {     ;
            this.acquireRead();
            while (iterator.hasNext())
```

```
                {
                  rA  = iterator.next();
                  if(naAG.equals(rA.getName()))
                      {
                            break;
                      }// end if
                }
          this.releaseRead();
          return (rA) ;
      }
      catch(Exception e)
      {
          System.err.println(e);
        return null ;
      }

}

      public void upadateStatus (String naAU , boolean bS )
      {
          AgentRecord rA = null ;
          // This method to update the status of agents in the list
          Iterator<AgentRecord> iterator = ListOfAgents.iterator();
          try
          {
                this.acquireWrite();
                while (iterator.hasNext())
                      {
                        rA  = iterator.next();
                        if(naAU.equals(rA.getName()))
                            {
                               rA.changeStatus(bS);
                              break;
                            }// end if
                      }
                this.releaseWrite();

          }
          catch(Exception e)
          {
                System.err.println(e);
          }

      }

      public void deleteAgent  (String naD)
      {

          AgentRecord rA ;
          // This method to update the status of agents in the list
          Iterator<AgentRecord> iterator = ListOfAgents.iterator();
          try
          {
                this.acquireWrite();
                while (iterator.hasNext())
                      {
                        rA  = iterator.next();
```

```
                        if(naD.equals(rA.getName()))
                            {
                              ListOfAgents.remove(rA);
                              break;
                            }// end if
                      } // end while s
                this.releaseWrite();

          }
          catch(Exception e)
          {
                System.err.println(e);
          }

    } // end deleteAgent method

    public ArrayList<AgentRecord>  searchAvailabelAgents ()
    {
          ArrayList<AgentRecord> x = new ArrayList<AgentRecord> ();
          AgentRecord rA = null  ;
          // This method to insert Information about new Agent
connected to the broker
          Iterator<AgentRecord> iterator = ListOfAgents.iterator();
          try
          {
                this.acquireRead();
                while (iterator.hasNext())
                      {
                        rA  = iterator.next();
                        if(rA.getStatus()) x.add(rA);

                      } // end While
                this.releaseRead();

                return x  ;

          }
          catch(Exception e)
          {
                System.err.println(e);
              return null ;
          }

    }

    public boolean  nameExists (String Na)
    {
          boolean isX = false ;
          AgentRecord rA = null  ;
          // This method to give all the age Information about new
Agent connected to the broker
          Iterator<AgentRecord> iterator = ListOfAgents.iterator();
          try
          {
                this.acquireRead();
                while (iterator.hasNext())
                      {
```

```
                        rA  = iterator.next();
                        if( rA.getName().equals(Na))
                        {
                              isX= true  ;
                              break ;
                        }

                      } // end While
                this.releaseRead();

                return isX ;

          }
          catch(Exception e)
          {
                System.err.println(e);
              return true ;
          }

    }

    public boolean  isAvailable (String Na)
    {
      AgentRecord rA = null ;
      boolean x =false ;
    // This method to insert Information about new Agent connected to
the broker
    Iterator<AgentRecord> iterator = ListOfAgents.iterator();
    try
    {
          this.acquireRead();
          while (iterator.hasNext())
                {
                  rA  = iterator.next();
                  if( rA.getName().equals(Na))
                  {
                      if (rA.getStatus()==true)
                            {
                              x= true ;
                            break ;
                            }
                  }

                } // end While
          this.releaseRead();


    }
    catch(Exception e)
    {
          System.err.println(e);
     }
    return x;
    }

  // grabs lock to read
```

```
	public synchronized void acquireRead() throws
	InterruptedException
	{
		while(writing || waitingW>0)
			{
			wait();
			}
		++readers;
	}

  // releases lock to read
	public synchronized void releaseRead() {
		--readers;
		if(readers==0)
		{
			notifyAll();
		}
	}

   // grabs lock to write
	// grabs lock to write
	public synchronized void acquireWrite() throws
InterruptedException{
		++waitingW;
		while(readers>0 || writing)
		{
		wait();
		}
		--waitingW;
		writing =true;
		}

	// releases lock to write
	public synchronized void releaseWrite()
		{
			writing=false;
			notifyAll();
		}

	// provides state information about the shared data object
/*		private synchronized void dumpState() {
		System.out.print ("[R="+readers+",S="+writing+"writing
reading,W="+waitingW+"]\n");
			}    */


}// end AgentsDetails Class
```

Agent Class (code 6):

```
/* the class is used to build an agent that is used to make a
communication between a
 broker (server)  and a user and besides it makes avalible  to an agent
to connect with another agent */

import java.io.*;
import java.net.*;
import java.util.*;


public class Agent
{

public static BufferedReader input = new BufferedReader(new
InputStreamReader(System.in));
public static final int choosingAgentStatus =1;
public static final int displayAgentsList =2;
public static final int AvailabeAgent  =3;
public static  int state = choosingAgentStatus ;
public static boolean  exit = false ; // shut doen the system
public static Vector<String>  LV ;
public static int portChat =4441;
public static String fServer =" ";
public static String  getUserName()
{
String UserN="";  // User name of client

try
{
do
{
System.out.print("Enter User Name: ");
UserN = input.readLine();
if (UserN.length()<3) System.out.print(" User Name is invalid , it
should be more than 3 letters");
}
while (UserN.equals("")|| UserN.length()<3);  // repeat until valid
user name is given
}
catch(IOException e)
{
System.err.println(e);
System.exit(1);
}

return (UserN);
}
public static String  getUserStateChoice()
{     String sc="";     //selecting any status that the agent want to

try
{
do
{
```

```
System.out.println("Enter What you want to be?  ");
System.out.println("Press (1): I want to talk   ");
System.out.println("Press (2): I want to listen  ");
System.out.println("Press (0): I want to shut down the application  ");
System.out.print("Choice: ");
sc = input.readLine();
if (!(sc.equals("1")||sc.equals("2")||sc.equals("0")))
{  //   inappropriate input
System.out.println("Incorrect choice , Enter Again");
sc="";
}
}while (sc.equals(""));  // repeat until valid user name is given

} catch(IOException e)
{
System.err.println(e);
System.exit(1);
}
return sc ;
} // end get choice

public static void main(String []args)
{

// ---------------------------------------------------------------------
-----------
int port=4444 ,          // server port
agentChoice  ;   // agentChoice

String sPort="",         // server port
ip="",                   // IP of server
userName="" ,      // User Name
agentIP="" ,       // selected agent IP
selectedAgent ,    // selected agent name
inToServ  ;
// Get ip, port & user name from the client

// A0 .
try
{

// get IP from the user
System.out.print("\n\nEnter IP of the server ( default IP 127.0.0.1):
");
ip = input.readLine();
if (ip.equals("")) ip = "127.0.0.1";      // default IP

// get port from user
System.out.print("Port Number (default port 4444): ");
sPort = input.readLine();
if (sPort.equals(""))
port = 4444;// default port

else port = Integer.parseInt(sPort);


// create a new socket
```

```
Socket socket = new Socket(ip, port);

// to get data to and from server
BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
PrintWriter pwr = new PrintWriter(socket.getOutputStream(), true);
// A0 .


// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// IP, port and username is complete at this point
// Now, create a socket to connect to server.
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


// Connection successfull at this point, so inform user about this
fServer =br.readLine();
System.out.println("You are "+fServer );
System.out.println("\n\n\t\tConnection is successful.\n\t\t------------
----------");
// A1 .
// ----------------------------------------------------------------
//     get user name from the client
userName = getUserName();
if ( userName.toLowerCase().equals("exit"))  exit=true ; // user type
exit to end its process
// send user name to the server
pwr.println(userName);

// A2.
// ----------------------------------------------------------------------

while ((fServer = br.readLine()) != null && !exit  ) //A.3
{

if (fServer.equals("exit"))
{
exit=true ;// shut down the process of agents
break ;
// A.5
}
else if ( fServer.equals("SR") )
{
// successfull registration on Server for the agent
System.out.println("\n\n\t\t The Registration is successful.\n\t\t-----
----------------------");
System.out.println("\n\n\t\t Your User Name is
"+userName+"\n\t\t++++++++++++++++++++++");

break;
//A.4
} // end if

else if (fServer.equals("USR"))
{
/*    Unsuccessfull registration on Server for the agent , enter again
new User Name, because
```

```
the user name that is entered before is duplicated , it is not allowed
on Server, so the agent
should enter again  */
System.out.println("\n\n\t\t The Registration is Unsuccessful because
User Name is duplicate .\n\t\t----------------------------------------
---------------");
System.out.println("\n\n\t\t Reinsert new User Name Again.\n\t\t-------
-------------------------\n");
userName = getUserName();
pwr.println(userName);
} // end if

}// end while


// ---------------------------------------------------------------------

//Give the agent a chance to choose its status (talking , listenning)
l1:   while (!exit)
{
// make user to take a decision about its state (talker or listener )
agentChoice = Integer.parseInt(getUserStateChoice()) ;
//A.6


l2:  switch ( agentChoice  )
{
case 0 : exit = true ;
break;
// A.30

case 1 :  // A.9
// talking state
// send to the broker to make the agent unavailable
pwr.println("talker");
//A.10
//send to the broker an order to bring him the list of available agents
do
{
System.out.print("\n\n Enter (3) to display available agent in the
network \n Enter(0) to shutdown ?\n ----> ");
inToServ = input.readLine();

if(!(inToServ.equals("3")||inToServ.equals("0")
))System.out.println("Wrong Input Try Again \n") ;
if (inToServ.equals("0")) { exit =true ; break l1;}
}  while (!(inToServ.equals("3")||inToServ.equals("0")));
pwr.println("DA");
//A.11
fServer ="";

// this While loop just of sign to inform the Agent the list will come
after that
w1: while ( (fServer = br.readLine()) != null)
{
if ( fServer.equals("ls") ) break w1;
else if ( fServer.equals("EMPTYLIST"))
```

```
{
System.out.println("There is no Available Agent , come later \n ") ;
// pwr.println("");
break l2 ;
}
else if (fServer.toLowerCase().equals("exit")) {exit= true ; break l1
;} // shuts down the process
}
// Start to print the list from Broker
fServer ="";
LV = new Vector<String> ();
// waiting the List of available agents
w2:  while (((fServer = br.readLine()) != null))
{

// print the list of agents
if (fServer.equals("le")) break w2 ;
System.out.println(fServer) ;
// save the List
LV.add(fServer);

}  // end while
// A.12

// While loop to ensure that the input By Agent contains in the List


do
{
System.out.println("\n Select the User Name ,  you would like to
contact or type (exit) to shutdown ");
selectedAgent = input.readLine();
if (
LV.contains(selectedAgent)||selectedAgent.toLowerCase().equals("exit"))
break;
System.out.println("\n You Enter Wrong Name  , Try Again ! ");

}while ( true  ) ;
// send Agent's choice to the broker and waiting
//A.15
if (selectedAgent.toLowerCase().equals("exit")){ exit =true ; break
l1;} // end the global loop
//A.16
pwr.println(selectedAgent);
// A.14
fServer ="";
agentIP="";
while ((fServer = br.readLine()) != null ) // waiting the IP of the
selected Agent
{
if (fServer.equals("NF"))
{
System.out.println("\n an Agent You have chosen becomes unavailable ");
break l1 ;
}
else {
agentIP=fServer;
```

```
break;
}
}
// A.17 ,
new ChatThread(portChat, agentIP,selectedAgent,userName,"t").start();
// A.18
break;


case 2 :pwr.println("listener");
//A.22
new ChatThread(portChat,userName,"l").start(); break l2 ;
//A.23

}// end switch


do{ } // this loop  wait until finish the chat
while (ChatThread.activeCount()==2);


// A.28
System.out.println("Do you want to Exit ?  (N)   : Y/N");
String endConnect = input.readLine();
if ( endConnect == null || endConnect.toLowerCase().equals("y")) break
l1;
//.A 29
}// end globale while loop

// send end connection
pwr.println("exit");
// A.30
}

catch(UnknownHostException e)
{   System.err.println("Don't know about host: "+ip);
System.exit(1);
}
catch (IOException e)
{
System.err.println("Couldn't get I/O for the connection to: "+ip+" :
"+port);
System.exit(1);
}


}     // End of main()
}     // End of class
```

Chat Thread Class (code 7) :

```
import java.io.*;
import java.net.*;

public class ChatThread extends Thread  {

protected ServerSocket  ss;
protected Socket  as ;
protected BufferedReader i;
protected PrintWriter o;
public String aIP , nameOtherAgent ,myName,  text ,inputb;
protected int chatPort ;
protected String St ;
public static BufferedReader input = new BufferedReader(new
InputStreamReader(System.in));


// constructor
public ChatThread (int chatPort,String myName,String status) throws
IOException
{

super("ThreadChat");
this.chatPort= chatPort ;
this.aIP = "";
this.nameOtherAgent="";
this.myName =myName ;
this.St=status ;


}
public ChatThread (int chatPort, String aIP ,String
nameOtherAgent,String myName,String status) throws IOException
{

super("ThreadChat");
this.chatPort= chatPort ;
this.aIP = aIP ;
this.nameOtherAgent= nameOtherAgent;
this.myName =myName ;
this.St=status ;


}

public boolean workAsServer()
{
try{
ss = new ServerSocket( this.chatPort);

//do
//{
System.out.println("waiting for connection.........");
```

```
as = ss.accept();
System.out.println(" connection with someone");

// System.out.print( as.getInetAddress().getHostAddress().toString());
//} while
(!as.getInetAddress().getHostAddress().toString().equals(this.aIP));

return (true);
} catch (IOException e)
{
System.err.println("Accept failed.");
return (false);
}

}
public boolean workAsClient()
{
try
{
as = new Socket (this.aIP,this.chatPort);
System.out.println(" make connection with "+this.nameOtherAgent);

return (true);
}  catch (IOException e)
{
System.err.println("Couldn't get I/O for the connection
to:"+this.nameOtherAgent);
return (false);
}
}

public  void run()
{
boolean x = false;
try{
if (this.St.equals("l"))
x = workAsServer() ; //work as server for the agent who want to listen
// A.18

else if ( this.St.equals("t"))
x=workAsClient(); //work as server for the agent who want to talk
//A.24

if(x == true) // make sure that everything working
{

o = new PrintWriter(as.getOutputStream(), true);
i = new BufferedReader(new InputStreamReader(as.getInputStream()));

System.out.println("\t\t\t +++++ ( Successful Connection
)+++++\t\t\t\n");


if (this.St.equals("t"))
{       // to make some type of synchronization amon chatting ,
//intial step one agent sends message as first message and the Other
should wait this message
```

```
System.out.print("|<"+this.myName+"> : ");
do { text =input.readLine();} while (text.equals(""));
o.println("|<"+this.myName+"> : "+text);   //A.19
while ((inputb=i.readLine())!= null ) //A.20
{// recieve message from the other Agent who is connect with you

System.out.println(inputb);
if ( inputb.toLowerCase().contains("end chat")) break; // A.21
// send the meesage to the other Agent who is connect with you
System.out.print("|<"+this.myName+"> : ");
do { text =input.readLine();} while (text.equals(""));
o.println("|<"+this.myName+"> : "+text);
if (text.toLowerCase().contains("end chat")) break ;

}
}
else if (this.St.equals("1"))
{

while ((inputb=i.readLine())!= null  )//A.25
{

// recieve message from the other Agent who is connect with you
System.out.println(inputb);
if ( inputb.toLowerCase().contains("end chat")) break; //A.21
// send the meesage to the other Agent who is connect with you
System.out.print("|<"+this.myName+"> : ");
do { text =input.readLine();} while (text.equals(""));
o.println("|<"+this.myName+"> : "+text); //A.26
if (text.toLowerCase().contains("end chat")) break ;

}// end while
ss.close();
}
o.close();
i.close();
as.close();
}// end if
else  System.out.println("\t\t++++++++ Error is encountered
++++++++\t\t");
}catch(Exception e)  { System.err.println(e) ; }
System.out.println("\t\t++++++++ End Session ++++++++\t\t");


}// end run
}// end ThreadChat class
```