Distributed Systems Assignment

RMI Chat Application

DT249/4

Lecturer: Martin McHugh

Pedro Mauricio Tavares D12123176

April 6, 2016

Contents

List of Figures

Chapter 1

Introduction

More complex than systems that run on a single processor, Distributed Systems (DS) involve several computers or components working together in a cooperating manner in order to achieve an objective. As a result, this will appear to its users as a single system [1] [2].

Chapter 2

Typical challenges in a DS

Because different parts of the system are running in different networks and usually have geographic differences, there are some issues and challenges that arise that need to be considered when designing Distributed Systems [3] [4].

2.1 Transparency

This is a very important challenge for Distributed Systems. The goal here is to not let the users perceive that the processes and resources of the system are physically distributed across multiple computers. If the Distributed System is able to present itself to the users as if it were only a single computer, we can say that the System is transparent. There are a few concepts of transparency such as Location, Relocation, Failure, migration and so on.

2.2 Openness

The System has to be designed around protocol standards, so that new components of hardware and software from different vendors can be integrated with existing components. The DS must adhere to standard rules. For example, a computer network consists of standard rules that drive and govern the format the way messages are sent and received. The DS have to follow these standards.

2.3 Scalability

Distributed Systems must remain effective in case there are an increased number of resources or users. Scalability is divided in three categories:

- Size: Adding more resources or users to the system;
- Geographically: Users and resources lie far apart;
- Administrative scalable: it remains easy to manage even if it consists of many different administrative organizations.

A downside about scalability is, a system that is scalable in one or more of these categories, often shows some loss of performance when it scales up.

The network linking the individual computers in the systems may also limit its scalability.

2.4 Security

This is a measure challenge in a Distributed System because it has a lot of components and they are as strong as they are weak. Not all machines within the networking system can be made physically secure. We have to be concerned with how to protect the systems assets which includes processing, storage, communication paths, user interfaces I/O as well as files and messages transmitted across along with many other objects.

2.5 Quality of Service

Here we have to be concerned with the quality of service that is delivered to system users, be specific in how we should implement the system in order to deliver a good quality of service to its users. Some non-functional properties can affect the quality of the services experienced by users, these properties are reliability, security and performance. Another very important aspect in order to enhance the quality of service is the adaptability to meet changing system configurations and resource availability.

2.6 Failure Management

A lot of things can fail in a Distributed System, for instance, any component can die and another component can get disconnected from the rest of the network and so on. Failure Management is the capacity of detecting contained failures within the system, and consequently repairing them without affecting other components. Managing failure is very difficult because of the fact that while some components fail, others continue to work. There are some techniques used to deal with failures, the most common are: Masking Failures: The ability of hiding detected failures or making them less severe; Tolerating Failures: Sometimes it is not feasible to try to detect and hide all failures, therefore, they need to be tolerated. For example, a web browser contacting a server that is down, we can't hide this failure, therefore, we need to show the error to the user.

2.7 Heterogeneity of platform

Heterogeneity of platform We also have to face the fact that Distributed Systems contain many different kinds of hardware and software working together in a cooperative fashion to solve problems. For Example: Networks are configured differently between big endian and little endian; Computer Hardware are built from different vendors; Computers have different Operating Systems; There are still legacy applications that exist, therefore the DS have to be able to accept communications with them; One first step to approach these challenges is to follow and adopt the main standards such as the Internet protocols for example.

2.8 Latency

Latency is the time it takes packets to flow from one part of the world to another. A remote invocation in a distributed system takes considerably more time than an invocation in a non-distributed system, this additional time delay must be taken into account. These challenges can be addressed when deciding how the data will be transmitted, chunks of data for example, if the users can receive the data partially, etc.

2.9 Concurrency

Concurrent execution arises naturally in Distributed Systems, particularly on the server side. Several clients will share the same resource in a Distributed System, having said that, it is likely that several clients might attempt to access the same resource at the same time. The process that manages the shared resource could just take one client at a time, but this would limit throughput. So, the services must allow multiple clients to share the same resource concurrently without interference between them. The access to resources must be synchronized.

Chapter 3

Implementation, Challenges and Problems

3.1 Challenges and Problems

3.1.1 Scalability

For this Chat Application, scalability is a challenge that needs to be faced. As we are aware, as a chat application becomes popular, the number of users drastically increases. We need to provide a way to scale the chat system.
The problem if we don't do this is the use of the system will be affected, increasing errors, the performance of the chat application would slow down as a very limited use of resources will not be enough to handle all users.

3.1.2 Security

This is a very common challenge in a chat Distributed System. The server must be shielded against malicious users in order to keep working as intended. On the client side, the UI has to be protected against malicious scripts (virus)that can affect the performance and functionality of the server or the other users. Another issue here is to address communication between users and make sure no outsider will intercept any conversation. If he does, he will be able to read all the conversations from the chat application, which will bring about data privacy issues.
Another problem that will occur if we don't address this issue is all the users in the chat being infected by viruses, even worse, the server could be infected as well.

3.1.3 Quality of service

This will involve a collection of things, including security for example. Some non-functional properties can affect the reliability and performance of the chat system and consequently affect the users. For example, if the user sends a message, this cannot take a long time to be delivered to the other users, furthermore, the system must be reliable and make sure that all messages are delivered.

If the quality of service problem is not addressed correctly, the users might get annoyed and not want to use the application because of the poor services.

3.1.4 Failure Management

In a chat application it is crucial that the server does not die, neither permanently nor temporally, because all the users connected to it will be affected. If the server goes down, messages that have been sent but not delivered can be lost, users might be disconnected from the chat and it will be an annoying problem when trying to connect the same users again in the conversation.
This is another example that will cause the users to abandon the application when this problem occurs. This is the kind of thing (users leaving) that application owners do not want.

3.1.5 Concurrency

More than one client of the chat system might need to access the same resource in the server at the same time, this is a very common thing when dealing with several users that are accessing the same set of resources in a short period of time. We have to make sure the server will be synchronized when handling these resource requests.
If the server does handle concurrency, a lot of users will have their services delayed. Again, this will be another reason for the users to leave and stop using the chat application.

3.2 Addressing the Challenges

In order to address the challenge of scalability, it will be necessary to build the application server in such a way that we can add more machines or resources in the future without changing anything in the code, building the system within an architecture layer which will make it easy to make changes, add or remove resources, on the server side. In addition, centralizing algorithms will not be a good idea as a considerable amount of messages will have to be routed over many lines. Another option is to monitor the number of users that are using the application and test the performance of the system, if the number of users are increasing and the performance is decreasing, we will know that is time to scale up the system.
Finally, hiding communication latency is a technique that can be done in order to achieve scalability, meaning that we have to try avoid waiting response service requests as much as possible [4].

The security challenge is more complex than the previous one, as we need to track and solve issues in the software as well as the hardware. There are four types of security issues to consider here:

- Interception: This happens when an unauthorized person gains access to the data being transmitted;
- Interruption: When a message is corrupted;
- Modification: When the data is modified and

- Fabrication: When some additional data or activity, that normally does not exist, is created [4].

For the security issues, we can use different techniques when implementing the chat application such as encrypting the data being transmitted so that a malicious person will not be able to read the contents of the message. Furthermore, creating methods of authentication and authorizing users to perform requests and get responses.

Quality of service issues can be addressed by increasing servers resources on pick times, meaning the time when the vast majority of users will be using the system. If we provide a server in the cloud, we can solve this issue easily as some cloud services provide us with these functionalities.
The application must provide a way to manage problems in case of failure. A solution would be, for instance, to keep a temporary buffer in the server with the last sent messages and the chat room saved, so that in case of failure, the users will know the last point before the server stopped.

The problem of concurrency has to be addressed within the code. The use of threads and some other Java commands will be used to solve problems brought with the lack of concurrency.

3.3 What makes implementation difficult

One of the things to consider when implementing this kind of application is the tool we are going to use. There are a few tools we can use such as Sockets or Java Remote Method Invocation (RMI) [5].
Sockets work more or less in the same way in any programming language like Java or Python, for example, the concepts are very similar. However, in this case, I will use Java RMI, which is a very powerful tool and we can achieve things that merely using sockets will be very difficult. Nonetheless, there is something that can make the implementation difficult, the fact that we have to learn about this new tool and make sure we become familiar with RMI architectures, how its layers work and ways to implement methods and so on.

Chapter 4

Evaluation

4.1 Measure the success of the system

In order to communicate only Strings are send from the client to the servers and the other way without a special meaning. The server distributes incoming strings to all clients without looking at the content. As a result of this evaluation the whole system is rebuild from scratch to have a system that uses the advantages of RMI.

The Chat Application will be measured in different circumstances. It will have a limited amount of users using it, so the application will be tested within different scenarios. Firstly, we will evaluate with half of the capacity, then as the time goes on, a new user will be added in order to use the system's resources. Eventually, we will run the system with the maximum amount of users it can support, checking if the performance of the chat application has changed.
Another kind of test that will take place is testing the chat application to run in different Operating Systems such Windows, Linux or both.

4.2 Strengths of the System

It will be hard to point in detail all the strengths of the system but I will highlight the main points below.Among others, primarily we can say that this system has:

- Good Degree of Transparency: This system will have a good degree of transparency as its users will not be able to tell where the physical its location is. In Addition, If I want to move the resources (server etc) to another location, the users will not be able to know it.
- Scalability: If the number of users increases , up to a certain amount of course, we will be able to scale up this system, for instance, we can add more resources to the server like more memory etc., without changing anything in its code structure.
- Concurrency: This system is built in a way that, coding perspective, it will allow users to access resources concurrently, so that, a user do not

need to wait until the server finishes processing the request from another user, but the both requests can be processed concurrently.

- Heterogeneity of platform: This systems is built on basic standards that allow communication flows from a specific type of hardware to a different type. The communication will happen regardless the vendor of specific computer. Furthermore, the application is built in a language that will allow users from different Operating system to communicate to one another. For example, a Linux OS can send message to a person using windows.

4.3 Weaknesses of the System

- Quality of service: The Quality of service of this chat application will not be the priority, because of the requirements and specifications the system was built in. This system will not be capable to send a message to any user that are not on-line, meaning that if certain user suddenly goes down, he will not be able to received the undelivered messages that were previously sent to him. Furthermore, old conversations cannot be retrieve as we didn't implement some kind of database to save the logs or conversations.

- Security: There is a very important point that needs to be made in relation to this system, its biggest weakness is security. One thing is there is no real authentication by the server side in order to allow a certain user to use it, any user who knows the IP address of the server will be able to connect and chat with others users [6]. This is not a commom feature in Big and well built distributed System, which security is a major concern [7]. Another thing to point to bring is, even if a malicious user is not able to find out the IP address of the server, he is still able to see the messages sent back in forth as the data is not encrypted. This could be solve by implement some functions in the code in order to encrypt the data though.

4.4 Under what conditions the system will work well

As long as the server is up and running, this specific System will work well in any condition. The reason for this is because of the limited amount of users that can use the system at same time. Nevertheless, as said before, if we want to scale up the system in order to let a bigger amount of users to use the system, for example, we have to re-test it in different conditions and start working on threads. See the evaluation topic.

4.5 Under what conditions the system will not work well

Everything here will depend on the server. If the communication between users and the server is not as expected, the system might not work well and have constant failures. All of this will depend on things that might affect the networking communication such as weather, networking traffic, nodes along the path, etc [8].

Chapter 5

System Design

5.1 Introduction

Communication is the process by data and information can be exchanged from individual to individual or from groups to individual and vice-versa through commom system [9]. Nowdays, Instant Messaging is an essential tool that is used world-wide, by individual or even companies. We have a few good examples that provide this services such as Facebook and Whatsapp [10].
The main purpose of this document is to describe the design of a Java Chat application, not so advanced as but similiar to IRC [11] that will allow users to connect to a centralized server in order to send text (string) messages across the networking to others users also connected.

5.1.1 In Scope

- The Application should be very simple to use, not requiring training;
- The users should have an option to connect and disconnect whenever they want to;
- The Users should communicating to each other only through text messages;
- The application should not allow users to send files to the chat room;
- The server must have an interface at which will make easier to the System Administrator to start and stop it.

5.1.2 Not in scope

- Save users conversations;
- Gather information about users;
- Monitor user's conversations

5.2 The System

The system will not be designed using sockets, but a higher level protocol called RMI [3]. The architecture is a client-server implementation [12], which consist of a central server with a number of clients that can connect to it in order to use the services. The code of the application will be implemented using the Object Oriented Approach, when everything will be treated as an object.
For methods that have to be used, a remote interface has to be defined for the client as well as the server. Then, another two classes, one for client and other for the server, must implement those methods. Further methods can be defined in the server and client class if needed.

5.2.1 Requirements

The server will be running constantly and waiting for a new user to connect in order to join the conversation with another clients. As soon as the client connects in the Chat application, a message is sent to the others clients, letting them know that a new client just joined the current chat.

Functional

- A client can connect and disconnect from the chat application as they wish;
- Clients can send messages in text format to all other users in the chat;
- A client must have an unique name in order to join the chat room;
- The server must keep a record of all clients connected in the chat room;
- A message have to be sent to the clients when someone join or leave the chat;

Non Functional

- A client should not wait more than 3 seconds in order to receive a message;
- The server must run constantly even with the maximum number of clients connected;
- System must be delivered before April 10th;

5.2.2 Development Environment

Netbeans [13] is a very popular Development Environment used for a vast amount of programmers, it will be used in order to build the chat application.

5.2.3 Assumptions and dependencies

This section highlight some assumptions and dependencies made while preparing this documentation and after some research of how a standard Chat application works.

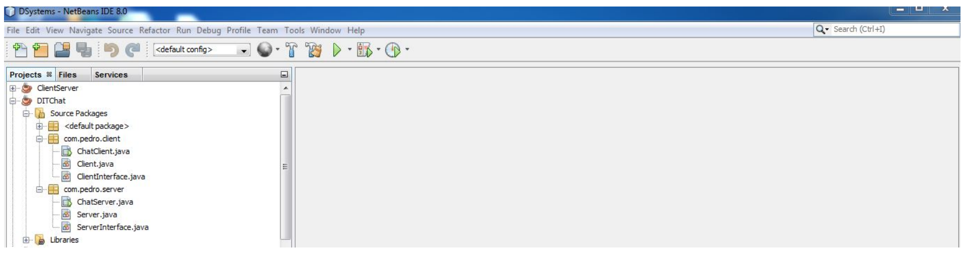


Figure 5.1: NetBeans Screenshot

- The Functionality of the client side interface will depend whether the server is working or not;
- As no conversations will be recorded, a Data Base design is no need;
- Is the entire responsibility of the client whatever they write and in the chat or what message they send to the other members during the conversations.

5.2.4 Design

Server and Client have to be two interfaces that implement the Remote interface. The interfaces are called ClientInterface and ServerInterface, the the classes that implement both are called Client.java and Server.java, both have to extend the UnicastRemotObject. The client has to connect to the server using the rmiregistry to get a remote reference of the server. It uses the servers authenticate method to connect and pass its own reference to the server, which will be the Client's name, hence the server can send back messages to the other clients using the send method of the Client, which is called tellMessage.
The System will have two more files called ChatClient and ChatServer. These files are for the user side, the first one to start the server and the latter to start the client.

A feature about the functionality is that the client side interface cannot invoke any methods from the server implementation directly but it needs the client implementation file.

RMI

Remote Method Invocation (RMI) is a Java provided mechanism that programmers can use to write code for object-oriented style so that objects on different computers interact in a network. It allows a Java object, which is executed on a Java Virtual Machine (JVM) to invoke a method of another Java object in another JVM.

Functionality

The key about the functionality that will make the chat between clients possible is that all people that are chatting to each other are part of the same group on the server.
In principle, the clients do not need to know anything what happens on the

server side, besides they do not need to know any information about the others clients as well. The server has a list of all participants, when someone sends a message, the client interface(ChatClient.java) will invoke a method from the client class (Client.java), which will call a method from the server class(Server.java), then the messages are passed back to the clients following the same path.
As we can see, the users will not know where the others users are located or where the server is placed (see the transparency section above).

Class Model

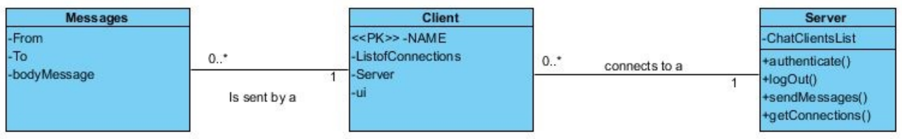


Figure 5.2: Class Model of the Chat

- A server can have zero or many clients connected to it;
- A client can connect to only one server;
- A client can send zero or many messages

For our current chat application, as there is no private communication between clients, the Message class does not need to be implemented. This will be a future solution and implementation.

System Architecture

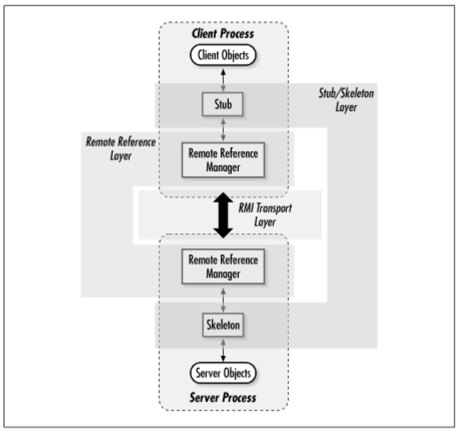


Figure 5.3: RMI Architecture[1]

[1]FigureSource: http://www.cs.ait.ac.th/~on/O/oreilly/java-ent/jenut/ch03_01.htm

System Model

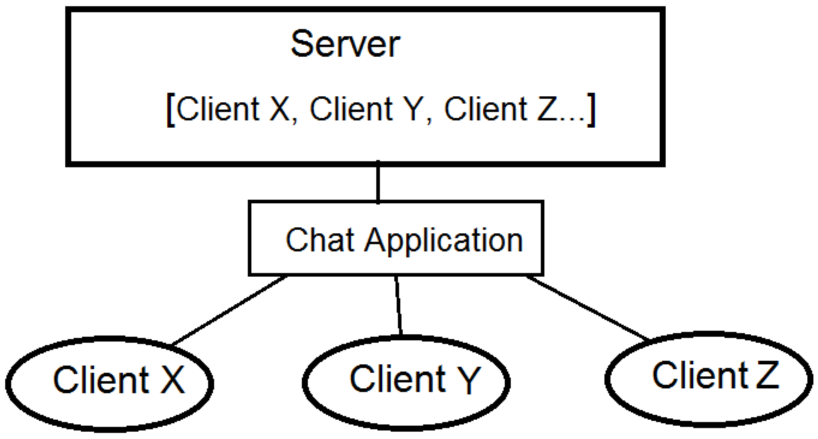


Figure 5.4: System Model

The Server will have a list of clients connected to the Application. This List will be frequently checked and updated very time a new client join the chat. The clients will be able to join the chat through the GUI application.

The Server

The server consist of three parts: The Interface (ServerInterface.java) , the server class (Server.class) and the GUI server (ChatServer.java).

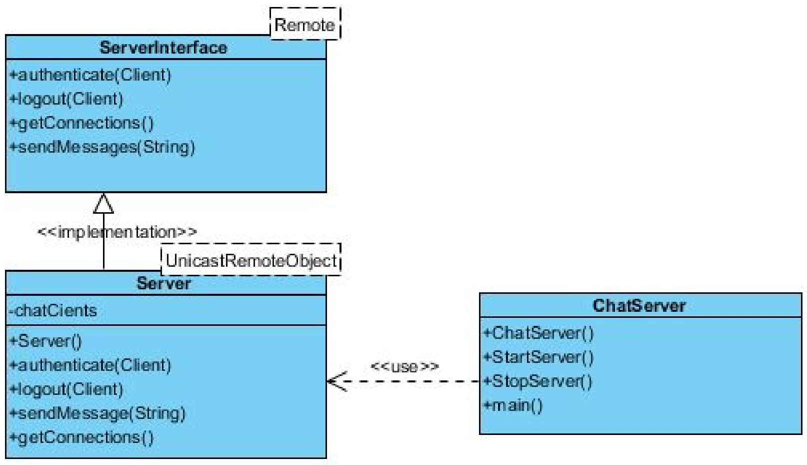


Figure 5.5: Server Class Diagram

As we can see, the server class inherent and implements methods from the interface class:

- Authenticate: This method will get a client as parameter and add into the chat room, updating the list of connected clients in the server;
- Logout: This method will be invoked when the client wants to leave the chat, once it is invoked, the clients list in the server will be updated again;
- GetConnections: This is a method that will tell who is connected to the other clients;
- SendMessage: This method is the method used to broadcast the message to every single client connected in the chat.

Some clients methods are triggered from the servers side such as getClientName and, which will get the name of the client as the method name self-explains, and tellMessage which is used to display messages to every single client in the chat.

Finally, we can see that the Server has a nice GUI interface so that the administrator will have a way to check quick whether the server is running or not. This interface also make easier to transfer the server from one machine to another, this might be the case if we want to scale up the system.

The Client

As we can see in figure 7.6, the client consist of Client Interface(ClientInterface.java), Client class(Client.java) and the client GUI (ChatClient.java).

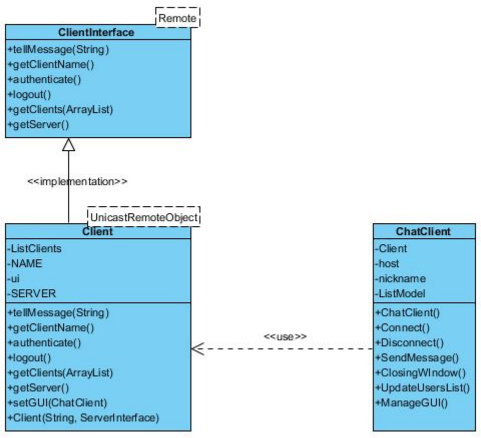


Figure 5.6: Client Class Diagram

The client interface provides methods that is implemented by the client class:

- TellMessage: Used also by the server, this methds display the message to every single client;

- GetClientName: Also used by the server, this simply gets the name of the client connected;
- Authenticate: This function is used to invoke the method Authenticate from the server in order to make the client join the chat and, therefore, let the server update its connected clients list;
- Logout: This method is the opposite of the authenticate, the client simply leaves the chat. For the server side, the method Logout is invoked in order to update the clients list;
- GetServer: Every client needs to have a server in order to call its methods, so this method just let the client class to invoke the servers method.

Besides all the implemented methods from the client interface, the client class still have another method called setGUI, which will set the User interface for each client.
As we can see, the logic is very well designed. Both Client and Server have their own interface, each at which will have some kind of hierarchy in order to invoke and access commands.
The Client user interface is well separated from the server interface and server methods. It can invoke the servers method directly but it needs the client class to do that. Only the client classe can invoke methods from the server class.

The Messages

For the purpose of this system, a message class(Message.java), which would have attributes such as "from", "body" etc, is not required, however, if, in the future, we want to add more functionalities, we have to develop such class and incorporate it into the current system.

The Test

In order to test the system, first of all we need to start the server, clicking on the start button. The default port the server will be running is 1099. As soon as we see the message in the server GUI saying that the server is running, we can start the client.

Normally, to start the client, we just need to put the server's ip address, if we are running in the same machine we can just leave it as it is (localhost), then we need to provide a unique name and click on the connect button, that's all.
We can open others clients and connect again and again until when we reach the maximum number of clients that can be connected in the chat.
Whenever a client tries to connect, the server interacts through its list to see if there is already a client registered with the same name, if so, the client will get a message, if not, the client will join the chat.

This program was tested on both Linux and Windows machines. When we use localhost, the application gives a faster response and the response was also good for remote machines. Latency of registration and un-registration cycle was checked and the program gave satisfactory results for all possible conditions.

5.2.5 Future Work

Some work in implementing the system still remains. This is a basic chat application which is left to be improved in the future. There are much more implementations we could do in the future in order to improve the system and make it more dynamic and robust such as:

- Change the clients list in the server side in order to store also a keyword session each client name;
- An authentication login and password could be implemented in order to enhance the security of the system;
- Implement a list on the server of different chat rooms that the clients want to join, so that we would have different clients chatting in different screens;
- Icons could be implemented on the client side so that the chat would be more interactive and "less boring";
- Implement a class called "message.java" with a few attributes in order to improve the functionality and scale up the features of the system;
- A data base for the chats and users can also be designed, conversations can be save, user-names and logins can be stored and so on;
- After implementing some of the previous examples, it would be possible to make the clients send private messages to each other user in the chat room. In addition, conversations can be saved and send even when the destination client is off-line;

As we can see, there are possibilities of the chat system be enhanced and even after the system is running, those methods previously mentioned can be added easily.

5.2.6 Conclusion

The objectives of this project is to show some of the methods of the Java platform that support networking communication and to design and implement a multi-client chat application for desktop computers which allows instant messaging by using the internet.

The method used here was RMI, which is a very good way to implement distributed system. The programmer doesn't have to care about the distribution during development, the objects can be distributed dependent on runtime.

The disadvantage of RMI is that it is only available for Java, however we can use other tools to build a client-server application such as CORBA [14].

Chapter 6

Conclusion

This assignment had the purpose to discuss some of the main characteristics and issues related to Distributed Systems, the main challenges were discussed and how they can affect the design, built and performance of a Distributed application.
We also showed how a simple chat application could be built and how would be its initial architecture. We also discussed how to implement and evaluate our chat application, what would be its specifics challenges.
Finally, this paper showed and presented a high level System Design of our application and discussed some of the technical things that we encounter on software Engineering process.

Chapter 7

Appendix

7.1 Use Case Diagram

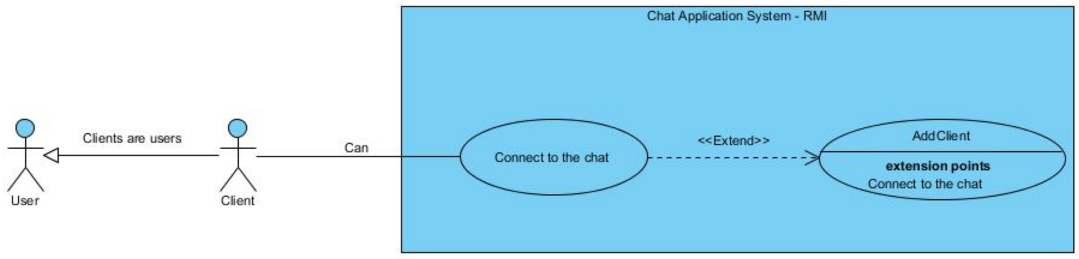
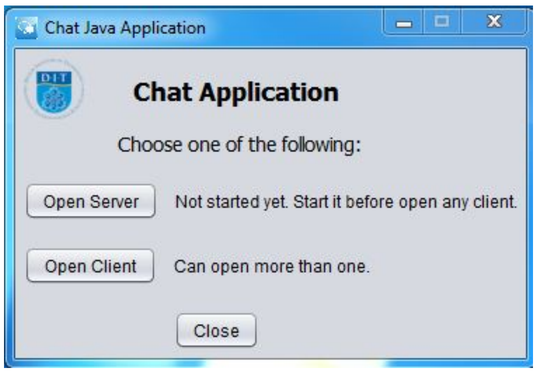


Figure 7.1: Use case Diagram

7.2 Start Program



Chat Application

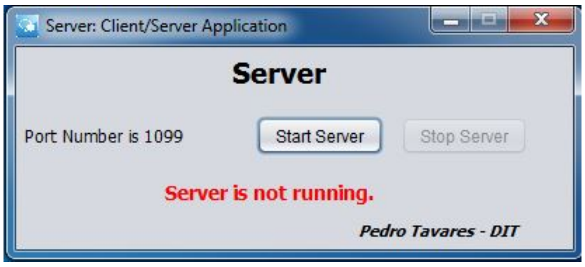
Choose one of the following:

Not started yet. Start it before open any client.

Can open more than one.

Figure 7.2: Main Program

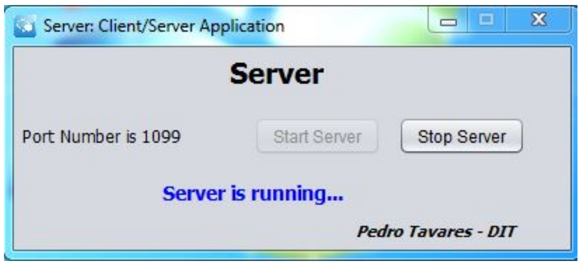
7.3 Server Start Program



Port Number is 1099

Server is not running.

Figure 7.3: Server Initial State

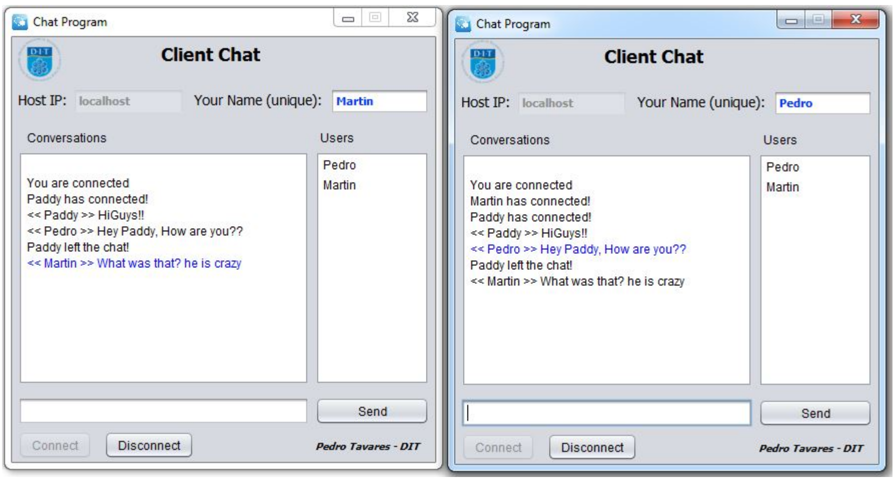


Port Number is 1099

Server is running...

Figure 7.4: Server running state

7.4 Client Start Program



Host IP: localhost Your Name (unique): Martin

You are connected
Paddy has connected!
<< Paddy >> HiGuys!!
<< Pedro >> Hey Paddy, How are you??
Paddy left the chat!
<< Martin >> What was that? he is crazy

Users: Pedro, Martin

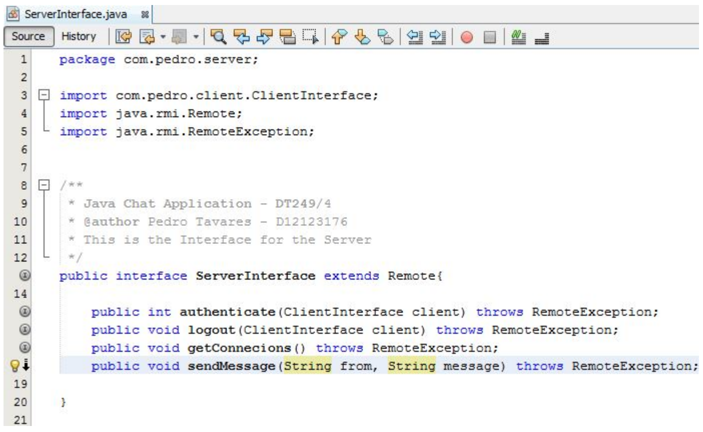
Host IP: localhost Your Name (unique): Pedro

You are connected
Martin has connected!
Paddy has connected!
<< Paddy >> HiGuys!!
<< Pedro >> Hey Paddy, How are you??
Paddy left the chat!
<< Martin >> What was that? he is crazy

Users: Pedro, Martin

Figure 7.5: Two Clients Running States

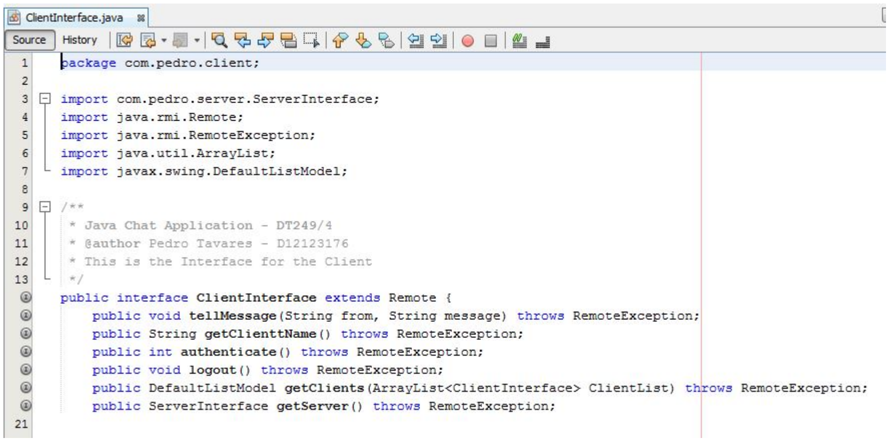
7.5 Server Interface Code



```java
package com.pedro.server;

import com.pedro.client.ClientInterface;
import java.rmi.Remote;
import java.rmi.RemoteException;


/**
 * Java Chat Application - DT249/4
 * @author Pedro Tavares - D12123176
 * This is the Interface for the Server
 */
public interface ServerInterface extends Remote{

    public int authenticate(ClientInterface client) throws RemoteException;
    public void logout(ClientInterface client) throws RemoteException;
    public void getConnecions() throws RemoteException;
    public void sendMessage(String from, String message) throws RemoteException;

}
```

Figure 7.6: Server Interface Java

7.6 Client Interface Code



```java
package com.pedro.client;

import com.pedro.server.ServerInterface;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;
import javax.swing.DefaultListModel;

/**
 * Java Chat Application - DT249/4
 * @author Pedro Tavares - D12123176
 * This is the Interface for the Client
 */
public interface ClientInterface extends Remote {
    public void tellMessage(String from, String message) throws RemoteException;
    public String getClienttName() throws RemoteException;
    public int authenticate() throws RemoteException;
    public void logout() throws RemoteException;
    public DefaultListModel getClients(ArrayList<ClientInterface> ClientList) throws RemoteException;
    public ServerInterface getServer() throws RemoteException;

```

Figure 7.7: Client Interface java

Bibliography

[1] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, “Distributed systems,” vol. 5, Jan. 2012.

[2] B. McCarty and L. Cassady-Dorion, “Java distributed objects,” vol. 1, Jan. 1999.

[3] R. Oechsle, “Parallele und verteilteanwendungen in java,” vol. 5, Jan. 2007.

[4] A. Tanenbaum and M. Steen, “Distributed systems,” vol. 2, Jan. 2007.

[5] Docs.oracle.com., “Trail: Rmi (the java tutorials).” http://docs.oracle.com/javase/tutorial/rmi/index.html, 2016. [Accessed 23 Mar. 2016].

[6] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, “Authentication in distributed systems: theory and practice,” vol. 5, pp. 265–310, Jan. 1992.

[7] M. Satyanarayanan, “Integrating security in a large distributed system,” vol. 1, pp. 247–280, Jan. 1989.

[8] K. Matsudaira, “Distributed systems basics handling failure: Fault tolerance and monitoring.” http://katemats.com/distributed-systems-basics-handling-failure-fault-tolerance-and-monitoring/, 2011. [Accessed 24 Mar. 2016].

[9] BusinessDictionary.com, “What is communication? definition and meaning.” http://www.businessdictionary.com/definition/communication.html, 2016. [Accessed 24 Mar. 2016].

[10] L. Zhang, “Building facebook messenger.” https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920, 2011. [Accessed 24 Mar. 2016].

[11] Irchelp.org, “Internet relay chat help.” http://www.irchelp.org/, 2016. [Accessed 24 Mar. 2016].

[12] SearchNetworking, “What is client/server (client/server model, client/server architecture)? - definition from whatis.com.” http://searchnetworking.techtarget.com/definition/client-server, 2016. [Accessed 24 Mar. 2016].

[13] Netbeans.org, “Welcome to netbeans.” https://netbeans.org/, 2016. [Accessed 24 Mar. 2016].

[14] Corba.org, “Corba web site!.” http://www.corba.org/, 2016. [Accessed 25 Mar. 2016].