SCREEN BROADCASTING USING RMI

(Remote Method Invocation)

A Report Submitted

In Partial Fulfilment of the Requirements

For the degree of

BACHELOR OF TECHNOLOGY

POOJA AGARWAL	**Roll No.:201410101110010**
ARUSHI TIWARI	**Roll No.:201410101110013**
CHANDRA SHEKHAR SINGH YADAV	**Roll No.: 201410101110017**
SHUBHANKAR RASTIOGI	**Roll No.: 201410101110040**

Guide: Ms. NEHA AGARWAL

FACULTY OF COMPUTER SCIENCE & ENGINEERING

INSTITUTE OF TECHNOLOGY,

SHRI RAMSWAROOP MEMORIAL UNIVERSITY

Dec, 2017

CERTIFICATE

It is certified that the work contained in the project report titled **SCREEN BROADCASTING USING RMI (Remote Method Invocation)**, by Chandra Shekhar Singh Yadav, Shubhankar Rastogi, Pooja Agarwal and Arushi Tiwari has been carried out under my supervision and that this work has not been submitted elsewhere for any other degree.

Ms. Neha Agarwal

Computer Science and Engineering Department

Shri Ramswaroop Memorial University

Dec,2017

Dr Shalini Agarwal

Computer Science and Engineering Department

Shri Ramswaroop Memorial University

ABSTRACT

The project Screen Broadcasting with RMI (Remote method Invocation) over a network.

The architecture used in the project is client - server architecture. The computer that broadcasts its screen works as server and the other connected computers are clients. A computer can be server or a client.

The computer that broadcasts the screen connected in a network captures its screen in the form of video and sends to the other connected computers in the same network.

The computers that receive the broadcast need to connect to the same network and then the connect to the broadcasting computer by specifying the IP address of the broadcasting computer.

The project uses JAVA RMI (Remote Method Invocation) API for the implementation of the remote connection for computers to the broadcasting computer.

ACKNOWLEDGEMENT

A research work owes its success from commencement to completion, to the people in love with researchers at various stages. Let me in this page express my gratitude to all those who helped us in various stage of this study.

I wish to express my sincere gratitude indebtedness to **Ms Neha Agarwal** (Department of Computer Science Engineering, SRMU) for introducing the present topic and for his inspiring guidance, constructive criticism and valuable suggestion throughout this project work.

I am also thankful to my Parents for their true help and inspiration. Last but not least, I pay my sincere thanks and gratitude to all the staff members and my project partner at **SRMU** for their support and for making our training valuable and fruitful.

Date:	Pooja Agarwal (201410101110010)
Place:	Arushi Tiwari (201410101110013)
	Chandra Shekhar Singh Yadav (201410101110017)
	Shubhankar Rastogi (201410101110040)

DECLARATION

We hereby declare that the project work entitled "**SCREEN BROADCASTING USING RMI(Remote Method Invocation)**" submitted to the Shri Ramswaroop Memorial University, is a record of original work done by us under the guidance of **Ms. Neha Agarwal**, Faculty of Computer Science and Engineering, Shri Ramswaroop Memorial University, Lucknow and this project work is submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of technology in Computer Science and Engineering. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree.

Signature:

Date:

TABLE OF CONTENT

TABLE OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

Have you ever felt the need for sharing the display contents of a computer with others, especially when all people cannot be assembled in a room due to physical constraint or lack of time? Video conferencing may be a good solution, but not so suitable when the flow of information is essentially one way – that too through some presentation slides. Display sharing over the network is an easy way to broadcast presentations on the network, so that other people can view the content without leaving the comfort of their own cabins.

The inspiration which leads the project is that it will very be helpful in daily lab classes and meeting as normally there are projectors installed to show the visual aspect of the implementations. But the content projected over a projector is not correctly visible to everyone as the one who is sitting in the last rows are not able to see what is actually happening because the projector cannot show everything equally from the first to last row.

That's why we need something that overcome this problem and make people see things more conveniently with even more ease. This can be done by simply sharing the screen to all the computers at same time who are connected over the same network. They simply need to connect to the computer who is about to broadcast and then they can easily see everything.

The project is about broadcasting a computer's (Laptop or Desktop) to the other systems connected over a network via wired or wireless connection. The computer who broadcasts, works as a server for all other systems who are connected in the network and are receiving the broadcast.

1.2 OBJECTIVE OF THE PROJECT

The objective of the project is to broadcast ones' computer screen to all the computers connected through the same network and this can be achieved by capturing the broadcaster's computer screen in the form of video and then sending to all the connected computers. This system works in client and server architecture. The broadcasting system works as server and the connected computers who are seeing the screen are the clients.

1.3 PURPOSE

The main purpose of our project is to share the display contents of a computer with others, especially when all people cannot be assembled in a room due to physical constraint or lack of time. Video conferencing may be a good solution, but not so suitable when the flow of information is essentially one way – that too through some presentation slides. Display sharing over the network is an easy way to broadcast presentations on the network, so that other people can view the content without leaving the comfort of their own cabins.

1.4 PROJECT SCOPE

1. Our project Screen Broadcasting will prove a good solution for virtual presentation meetings. As one not to be physically present in the presentation room and this will add comforts to the lives.
2. The project will become a good solution for presentation based classes and Labs.

CHAPTER 2
JAVA RMI

2.1 INTRODUCTION

RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package java.rmi.

The **Java Remote Method Invocation (Java RMI)** is a Java API that performs remote method invocation, the object-oriented equivalent of Remote procedure calls (RPC), with support for direct transfer of serialized Java classes and distributed garbage-collection.

The original implementation depends on Java Virtual Machine (JVM) class-representation mechanisms and it thus only supports making calls from one JVM to another. The protocol underlying this Java-only implementation is known as **Java Remote Method Protocol (JRMP)**.

2.2 ARCHITECTURE OF RMI

In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

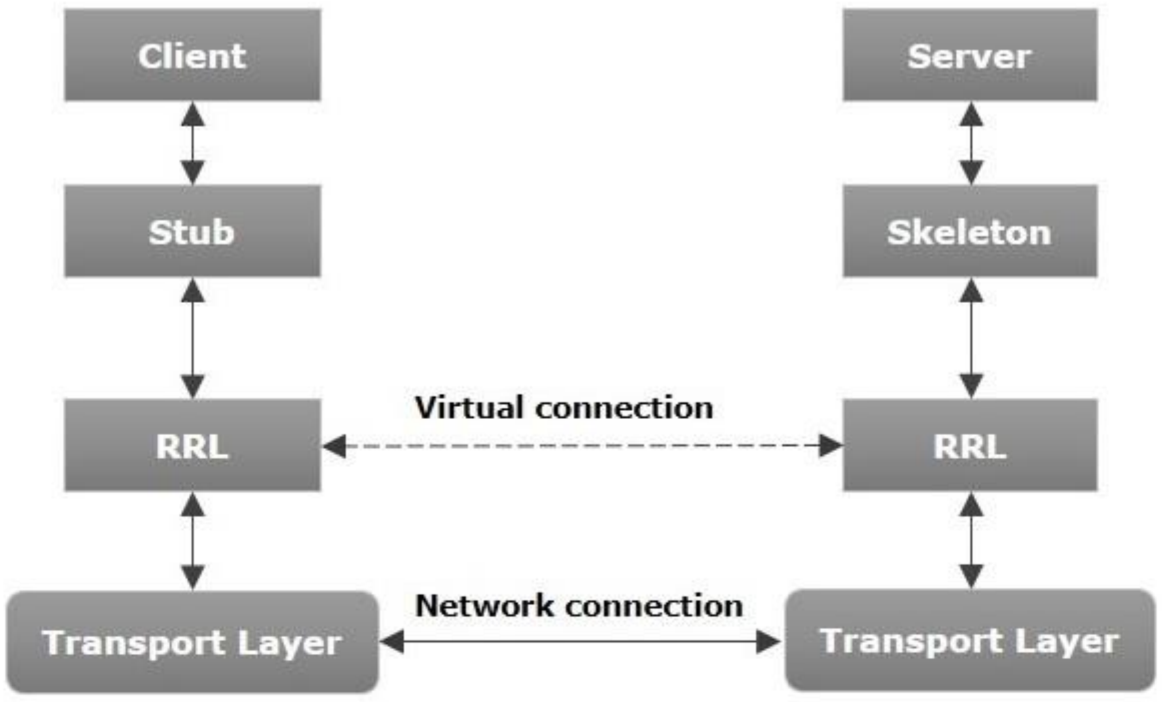


Figure 2.2.1 Architecture of RMI Application

The Components of RMI are as follows-

- ***Transport Layer*** − This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- ***Stub*** − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- ***Skeleton*** − This is the object which resides on the server side. Stub communicates with this skeleton to pass request to the remote object.
- ***RRL (Remote Reference Layer)*** − It is the layer which manages the references made by the client to the remote object.

2.3 <u>WORKING OF RMI APPLICATION</u>

The following points summarize how an RMI application works −

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called invoke() of the object remoteRef. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

2.4 UNDERSTANDING STUB AND SKELETON

RMI uses stub and skeleton object for communication with the remote object.

A remote object is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

2.4.1 stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads (unmarshals) the return value or exception, and
- It finally, returns the value to the caller.

2.4.2 skeleton

The skeleton is an object, acts as a gateway for the server-side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and
- It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, a stub protocol was introduced that eliminates the need for skeletons.

2.5 REQUIREMENT FOR DISTRIBUTED APPLICATION

If any application performs these tasks, it can be distributed application.

- The application need to locate the remote method
- It need to provide the communication with the remote objects, and
- The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

2.6 JAVA RMI EXAMPLE

The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

2.7 RMI EXAMPLE

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interact with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.

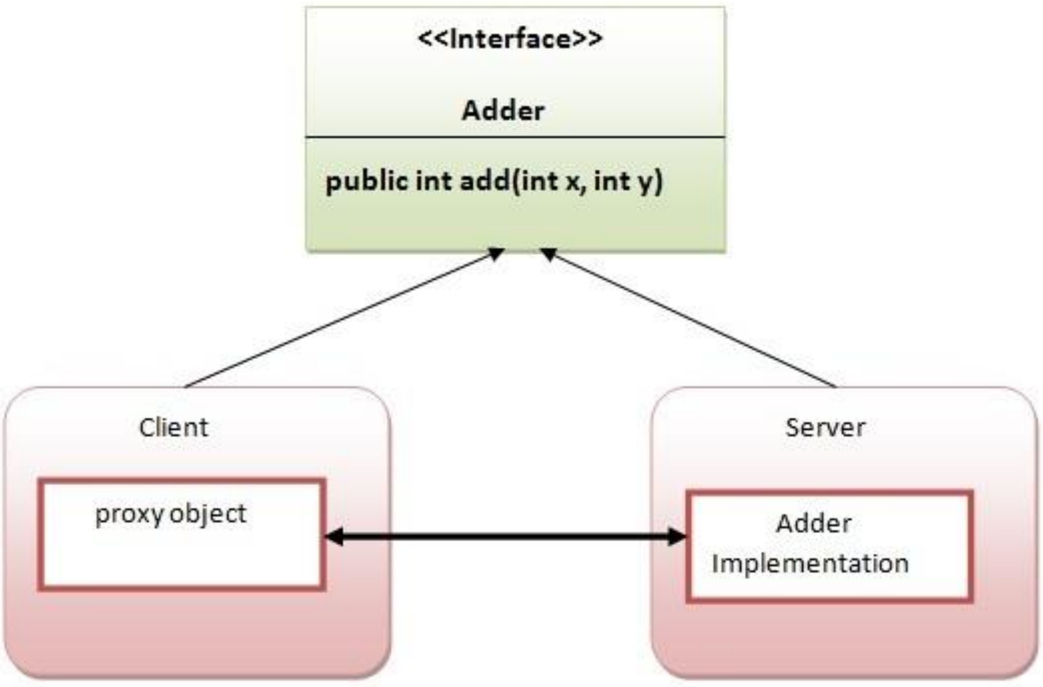


Figure 2.7.1 Implementation of RMI Example

1) Create the Remote Interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
```

2) Provide the implementation of the Remote Interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to either extend the UnicastRemoteObject class, or use the exportObject() method of the UnicastRemoteObject class.

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

3) Create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

5) Create and run the server application

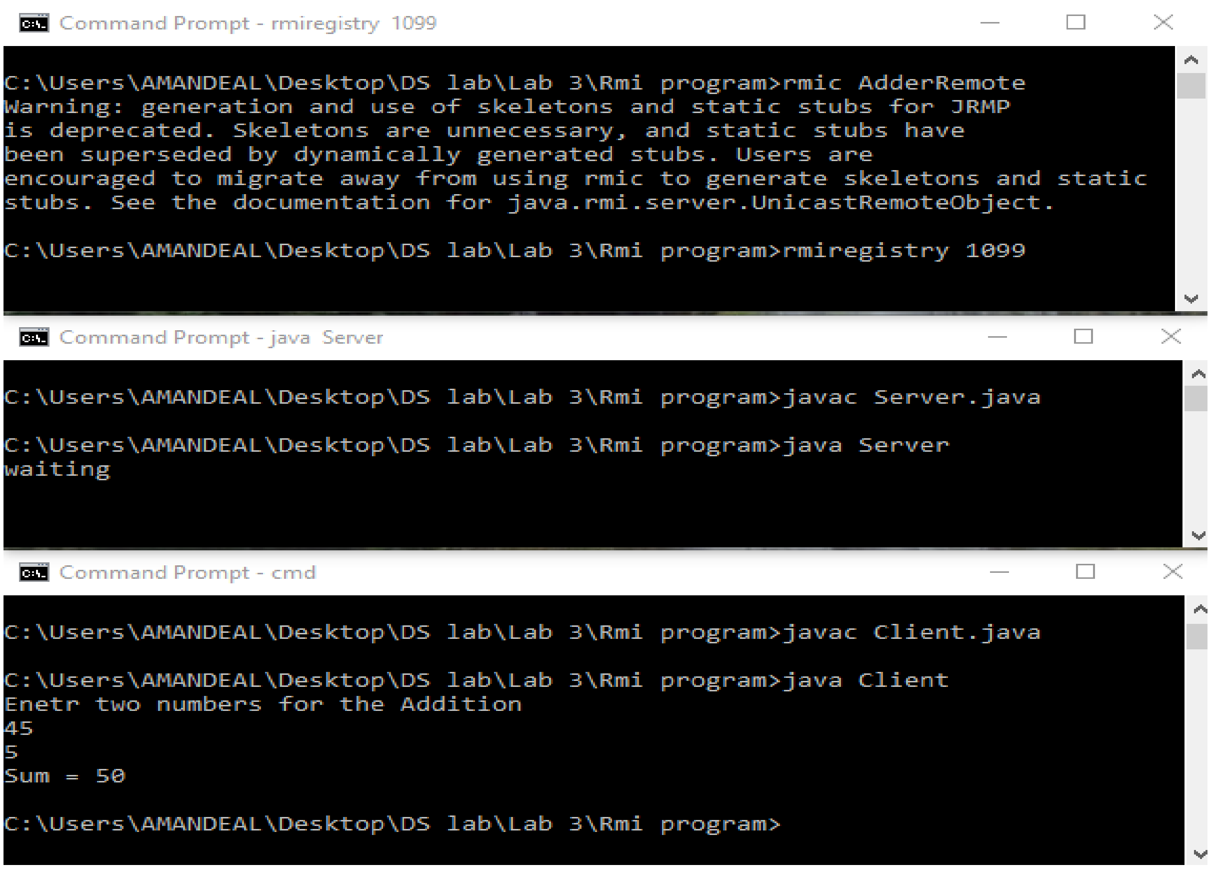
Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```



```
C:\Users\AMANDEAL\Desktop\DS lab\Lab 3\Rmi program>rmic AdderRemote
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

C:\Users\AMANDEAL\Desktop\DS lab\Lab 3\Rmi program>rmiregistry 1099
```

```
C:\Users\AMANDEAL\Desktop\DS lab\Lab 3\Rmi program>javac Server.java

C:\Users\AMANDEAL\Desktop\DS lab\Lab 3\Rmi program>java Server
waiting
```

```
C:\Users\AMANDEAL\Desktop\DS lab\Lab 3\Rmi program>javac Client.java

C:\Users\AMANDEAL\Desktop\DS lab\Lab 3\Rmi program>java Client
Enetr two numbers for the Addition
45
5
Sum = 50

C:\Users\AMANDEAL\Desktop\DS lab\Lab 3\Rmi program>
```

Figure 2.7.2 Output of the RMI Example

2.8 MARSHALLING AND UNMARSHALLING

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as **marshalling**.

At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as **unmarshalling**.

2.9 RMI REGISTRY

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using bind() or reBind() methods). These are registered using a unique name known as bind name.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using lookup() method).

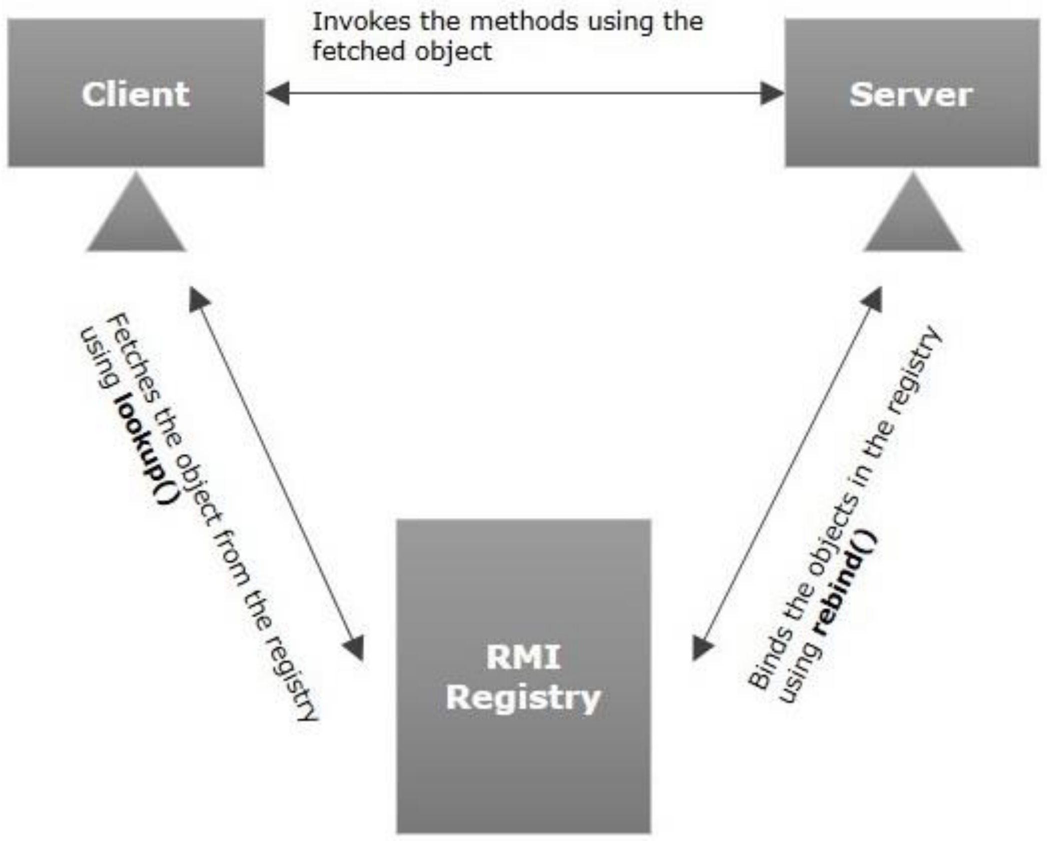

Invokes the methods using the fetched object
Client
Server
Fetches the object from the registry using lookup()
Binds the objects in the registry using rebind()
RMI Registry

Figure 2.9.1 Process of RMI Registry

CHAPTER 3
FLOWCHART

3.1 FLOW DIAGRAM

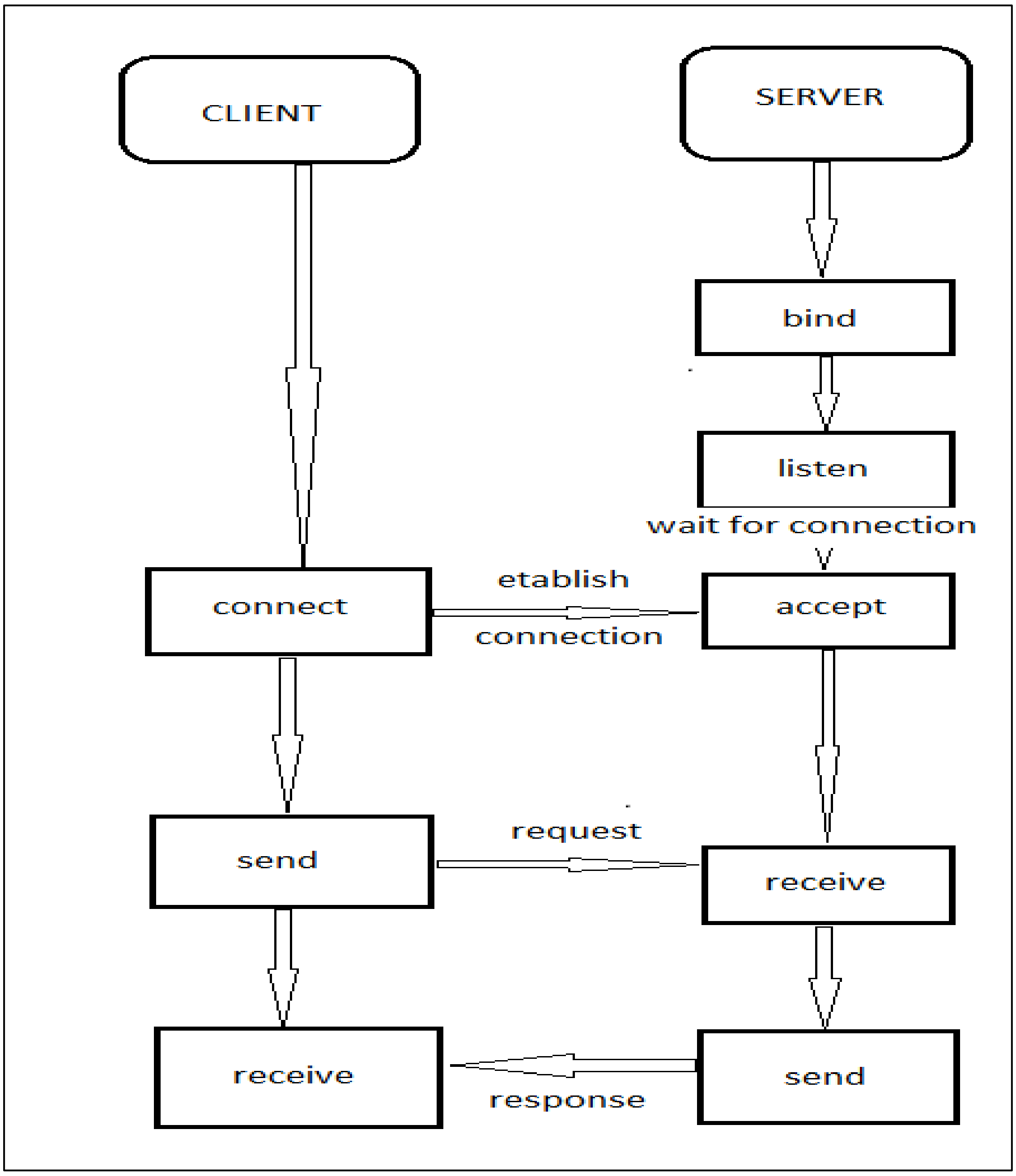


Figure 3.1.1 Flow Diagram of the Screen Broadcasting Process

3.2 STEPS INVOLVED IN SCREEN BROADCASTING PROCESS

1) Initialize the server and wait for connection

2) Start listening: start the registry for listening the server is ready to take request from the clients
3) Wait for connection: now server waits for the request from the clients
4) Open the client site enter the IP of the server to connect and wait for the response.
5) Server receive the requests and the in response broadcasts its screen to the connected client
6) The client receives the response able to see the screen of the server from remote location

CHAPTER 4
SOFTWARE AND HARDWARE REQUIREMENTS

4.1 PLATFORMS USED

- Eclipse
- Java Development kit
- Libraries Used: RMI-IIOP, RMIIO

4.1.1 RMI-IIOP

- It is read as "RMI over IIOP".
- It denotes the Java Remote Method Invocation (RMI) interface over the Internet Inter-Orb Protocol (IIOP), which delivers CORBA distributed computing capabilities to the Java platform.
- RMI-IIOP allows developers to pass any Java object between application components either by reference or by value.
- IIOP eases legacy application and platform integration by allowing the application components written in any CORBA supported languages to communicate with components running on the Java platform.
- Initially it was based on two specifications:
- Java Language Mapping to OMG IDL (Where OMG is "Object Management Group" and IDL is "Interface Definition Language").
- CORBA/IIOP
- With this inherited feature of CORBA, it supports multiple platforms and can make remote procedure calls to execute, subroutines on another computer as defined by RMI.

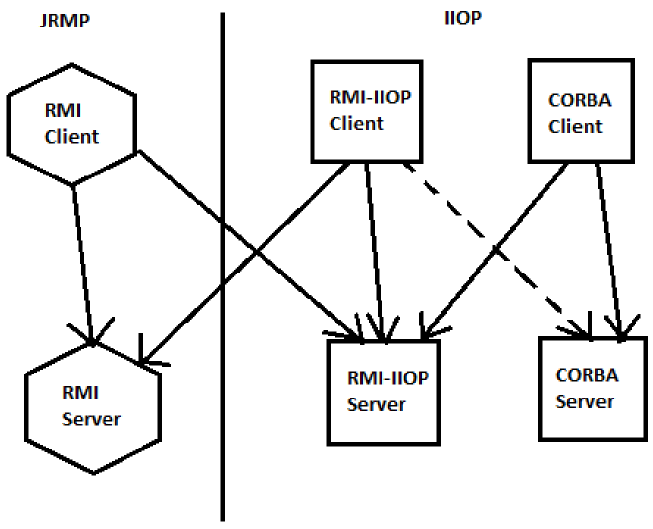


Figure 4.1.1 Introduction to RMI-IIOP

In Fig:4.1.1 above,

- The left part is representing the RMI (JRMP) model, the middle part the RMI-IIOP model, and the right part represents the CORBA model.
- Arrows are representing a situation in which a client can call a server. RMI-IIOP belongs in the IIOP world below the vertical line.
- The diagonal arrows crossing the border between the JRMP world and the IIOP world, implies that an RMI (JRMP) client can call an RMI-IIOP server, and vice versa.
- RMI-IIOP supports both JRMP and IIOP protocols.

4.1.2 RMIIO

RMIIO is a library which makes it as simple as possible to stream large amounts of data using the RMI framework (or any RPC framework for that matter). Who needs this? Well, if you have ever needed to send a file from an RMI client to an RMI server, you have faced this problem. And, if you did manage to implement a basic solution, it probably threw an OutOfMemoryError the first time someone tried to send a 2GB file. Due to the design of RMI, this common and deceptively simple problem is actually quite difficult to solve in an efficient and robust manner.

The RMI framework makes it very easy to implement remote communication between java programs. It takes a very difficult problem (remote communication) and presents a fairly easy to use solution. However, the RMI framework is designed around sending and receiving groups of objects which are all immediately available in memory.

The Features of RMIIO are as follows –

- Remote input and output streams

- Remote Iterator implementation for streaming collections of objects (SerialRemoteIteratorServer)
- Optional GZIP compression over the wire (input and output streams)
- Stream progress monitoring hooks (RemoteStreamMonitor)
- Optional low-latency streaming (noDelay)
- Serializable InputStream and OutputStream wrappers for remote input and output streams
- Pluggable RPC integration which can be used to integrate with frameworks other than RMI (see the RemoteStreamExporter section below for details)
- Utilities to facilitate robust RMI usage (RemoteRetry)

4.2 HARDWARE REQUIREMENT

- Windows 10,8.1,7
- 4–6 GB for a typical installation.
- Any Intel or AMD x86-64 processor
- No specific graphics card is required

REFERENCES

[1] https://en.wikipedia.org/wiki/Java_remote_method_invocation
[2] https://www.javatpoint.com/RMI
[3] http://openhms.sourceforge.net/rmiio/class_reference.html
[4] http://developeriq.in/articles/2011/jul/09/sharing-your-display-over-network-with-java/
[5] https://docs.oracle.com/javase/tutorial/rmi/overview.html
[6] http://openhms.sourceforge.net/rmiio/apidocs/index.html
[7] https://books.google.co.in/books?id=03tHa0FoUDMC&pg=PA157&dq=rmi&hl=en&sa=X&ved=0ahUKEwiehqjh0JDYAhUWT48KHeGMArgQ6AEIQTAF#v=onepage&q&f=false
[8] https://books.google.co.in/books?id=ORNjAQAAQBAJ&pg=PA39&dq=rmi&hl=en&sa=X&ved=0ahUKEwiehqjh0JDYAhUWT48KHeGMArgQ6AEIOzAE#v=onepage&q=rmi&f=false
[9] https://docs.oracle.com/javase/8/docs/technotes/guides/rmi-iiop/rmiiiopUsing.html