GOVERNMENT COLLEGE OF ENGINEERING, JALGAON

(An Autonomous Institute of Government of Maharashtra)

Department of Computer Engineering

Name: Vaibhav Bharat Sontakke	PRN No: 1942207
Class: T. Y. B. Tech.	Batch: L4
Date of Performance: _____	Date of Completion: _____
Subject: DSL	Sign. of Teacher with Date: _____

Experiment No.: 01

Aim: Program for Distributed Application using RMI

Title: Design a distributed application using RMI for remote computation where client submits two strings to the server and server returns concatenation of given strings.

Software Requirements: Ubuntu OS, Eclipse IDE 4.11 **Theory :**

Objective: The main objective of RMI is to provide the facility of invoking methods on the server. This is done by creating a RMI Client and RMI Server. RMI Client invokes a method defined on the RMI Server.

Remote Method Invocation (RMI) is an API which allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, an object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side). RMI creates a public remote server object that enables client and server side communications through simple method calls on the server object.

Working of RMI

The communication between client and server is handled by using two intermediate objects: Stub object (on client side) and Skeleton object (on server side).

1. **Stub Object**

The stub object on the client machine builds an information block and sends this information to the server. The block consists of

- An identifier of the remote object to be used
- Method name which is to be invoked
- Parameters to the remote JVM

2. **Skeleton Object**

The skeleton object passes the request from the stub object to the remote object. It performs following tasks

- It calls the desired method on the real object present on the server.
- It forwards the parameters received from the stub object to the method.

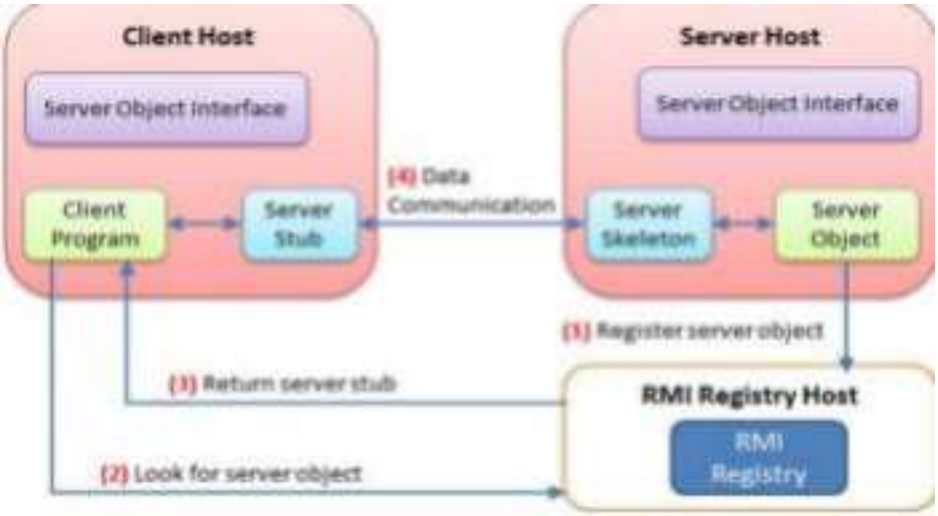

Figure: Working of RMI

Steps to Implement Interface

1. Defining a remote interface
2. Implementing the remote interface
3. Creating Stub and Skeleton objects from the implementation class using rmic (rmi complier)
4. Start the rmiregistry
5. Create and execute the server application program
6. Create and execute the client application program.

Step 1: Defining the remote interface

The first thing to do is to create an interface which will provide the description of the methods that can be invoked by remote clients. This interface should extend the Remote interface and the method prototype within the interface should throw the RemoteException.

```
// Creating a Search interface import
java.rmi.*;
public interface Search extends Remote
{
  // Declaring the method prototype public String query(String
  search) throws RemoteException;
}
```

Step 2: Implementing the remote interface

The next step is to implement the remote interface. To implement the remote interface, the class should extend to the UnicastRemoteObject class of java.rmi package. Also, a default constructor needs to be created to throw the java.rmi.RemoteException from its parent constructor in class.

```
// Java program to implement the Search interface
import java.rmi.*; import java.rmi.server.*;
public class SearchQuery extends UnicastRemoteObject implements
                Search
```

```
{
  // Default constructor to throw RemoteException
  // from its parent constructor
   SearchQuery() throws RemoteException
   {
      super();
   }

  // Implementation of the query interface public
  String query(String search)
                  throws RemoteException
   {
      String result;
      if   (search.equals("Reflection    in    Java"))
         result = "Found";
      else
         result = "Not Found";

      return result;
   }
}
```

Step 3: Creating Stub and Skeleton objects from the implementation class using rmic The rmic tool is used to invoke the rmi compiler that creates the Stub and Skeleton objects. Its prototype is rmic classname. For above program the following command need to be executed at the command prompt rmic SearchQuery

Step 4: Start the rmiregistry

Start the registry service by issuing the following command at the command prompt start rmiregistry

Step 5: Create and execute the server application program

The next step is to create the server application program and execute it on a separate command prompt. The server program uses createRegistry method of LocateRegistry class to create rmi registry within the server JVM with the port number passed as argument.

The rebind method of the Naming class is used to bind the remote object to the new name.

```
//program for server
application import java.rmi.*;
import java.rmi.registry.*;
public class SearchServer
{ public static void main(String args[])
   { try
      {
        // Create an object of the interface
        // implementation class
         Search obj = new SearchQuery();

        // rmiregistry within the server JVM with
        // port number 1900
         LocateRegistry.createRegistry(1900);

        // Binds the remote object by the name
```

```
            // geeksforgeeks
             Naming.rebind("rmi://localhost:1900"+
                        "/geeksforgeeks",obj);
                        }
            catch(Exception ae)
            {
               System.out.println(ae);
            }
        }
    }
```

Step 6: Create and execute the client application program

The last step is to create the client application program and execute it on a separate command prompt .The lookup method of the Naming class is used to get the reference of the Stub object.

```
//program for client
application import java.rmi.*;
public class ClientRequest
{ public static void main(String args[])
   {
      String  answer,value="Reflection  in  Java";
      try
      {
        // lookup method to find reference of remote object Search
         access =
            (Search)Naming.lookup("rmi://localhost:1900"+
                              "/geeksforgeeks");
         answer = access.query(value);
         System.out.println("Article on " + value +
                     " " + answer" at GeeksforGeeks");
      }
      catch(Exception ae)
      {
         System.out.println(ae);
      }
   }
}
```

Algorithm:

Steps:

1. Define a Remote Interface that contains methods which can be remotely invoked on remote object. Each Remote Interface have to extend – Java.rmi.Remote

2. Interface and each remote method declared in the remote interface have to throw java.rmi.RemoteException. **Listing 1**: Define Remote Interface, declaring Remote Method. **Listing 2**: Defining class that extends and implements UnicastRemoteObject and Interface

3. Generate Stub & Skeleton classes using rmic tool:
   rmic <remote-class>

4. compile and generate Stub and Skeleton Classes 5. Start Registry Services using rmiRegistry tool:

rmiRegistry <port-number> or **rmiRegistry**

- Create an object of the RMI server and register its stub in the RMI registry. Java.rmi.Naming class provides a static method for registration in the lookup of rmiStub.

- **bind()** : This method is used to register a remote object (stub of object) in the rmiRegistry. Syntax: public static void bind(String name, Remote object) throws RemoteException, AlreadyBindException

- **rebin()**: This method is used to rebind the remote object in the rmiRegistry.

Syntax: Public static void rebind(String name, Remote object) throws RemoteException **Lookup()** : This method is used to lookup remote stub in the rmi Registry. Syntax: Public static Remote lookup(String name) throws RemoteException

Listing 4: Define a RMI Server Class

6. Now define a RMI Client. A RMI Client has to obtain the reference of remote stub from the RMIregistry and invokes the remote method.

Listing 5: Define a RMI Client Class

If RMI registry is running on non-default port on a different machine, then following steps are required for registration:

Java.rmi.registry.LocateRegistry class is used to create a registry object. This class provide following methods:

Public static Registry getRegistry (int port);

Public static Registry getRegistry(String host, int port);

7. **Listing 6**: Define how to register a server in Registry if RMI Registry run on non-default port Info regarding RMI registry is provided to the lookup () method through "Lookup String". Lookup string has the following format:

LookupString:

"protocol:hostname:portNo/registeredNameserver"

8. **Listing 7**: If RMI Registry run on non-default port, define a RMI Client

Compile RMI Client and Server Classes and Interface

- First of all compile all the classes and interface using javac command: javac *.java Now generate stub class file from remote method class that implements Remote Methods using rmic tool for remote : rmic hello
- Now start rmiregistry in one console:

- Start rmiregistry in command window using command rmiregistry

- We can also rmiregistry on different ports using rmiregistry <port-no>.
- Star RMI server using java <server-class-name> command Using Java <rmi-server-class-name> & is placed in the command window to start rmi server.

Now start a RMI client using java <client-class-name> and command line arguments.

Conclusion: In this Practical, we learnt about development of distributed applications using RMI.

……

Mrs. Archana Chitte

Name & Sign of Course Teacher