**Experiment 2 – RPC/RMI**

**Learning Objective:** Student should be able to Built a Program for Client/server using RPC/RMI

**Tools :**Java

**Theory:**

**Remote Procedure call**

A remote procedure call (RPC) is an inter-process communication that allows a computer program to cause a procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.

It further aims at hiding most of the intricacies of message RPC allows programs to call procedures located on other machines. But the procedures ‘send’ and ‘receive’ do not conceal the communication which leads to achieving access transparency in distributed systems.

Example: when process A calls a procedure on B, the calling process on A is suspended and the execution of the called procedure takes place. (PS: function, method, procedure difference, stub, 5 state process model definition)Information can be transported in the form of parameters and can come back in procedure result. No message passing is visible to the programmer. As calling and called procedures exist on different machines, they execute in different address spaces, the parameters and result should be identical and if machines crash during communication, it causes problems.

3.1.1 **RPC Operations:**

1) **Conventional procedure call**

For a call of a program, an empty stack is present to make the call, the caller pushes the parameters onto the stack (last one first order). After the read has finished running, it puts the return values in a register and removes the return address and transfers controls back to the caller. Parameters can be called by value or reference.

**Call by Value:** Here the parameters are copied into the stack. The value parameter is just an initialized local variable. The called procedure may modify the variable, but such changes do not affect the original value at the calling side.

**Call by reference:** It is a pointer to the variable. In the call to Read, the second parameter is a reference parameter. It does not modify the array in the calling procedure.

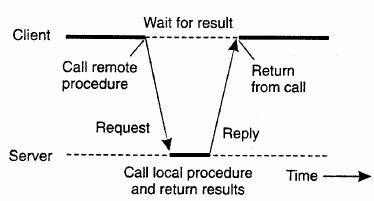
**Call-by-copy:** Another parameter passing mechanism exists along with the above two, its called call-by-copy or Restore.

Here the caller copies the variable to the stack and then copies the variable to the stack and then copies it back after the call, overwriting the caller’s original values. The decision of which parameter passing mechanism to use is normally made by the language designers and is a fixed property of the language. Sometimes it depends on the data type being passed.

2) Client and Server Stubs

1. A stub in distributed computing is a piece of code used for converting parameters passed during a Remote Procedure Call.
2. The main idea of an RPC is to allow a local computer (client) to remotely call procedures on a remote computer (server).
3. The client and server use different address spaces, so conversion of parameters used in a function call have to be performed; otherwise the values of those parameters could not be used, because of pointers to the computer's memory pointing to different data on each machine.
4. The client and server may also use different data representations even for simple parameters. Stubs are used to perform the conversion of the parameters, so a Remote Function Call looks like a local function call for the remote computer.

For transparency of RPC, the calling procedure should not know that the called procedure is executing on a different machine.



**Client Stub:** Used when read is a remote procedure. Client stub is put into a library and is called using a calling sequence. It calls for the local operating system. It does not ask for the local operating system to give data, it asks the server and then blocks itself till the reply comes.

**Server Stub:** when a message arrives, it directly goes to the server stub. Server stub has the same functions as the client stub. The stub here unpacks the parameters from the message and then calls the server procedure in the usual way.

**Summary of the process:**

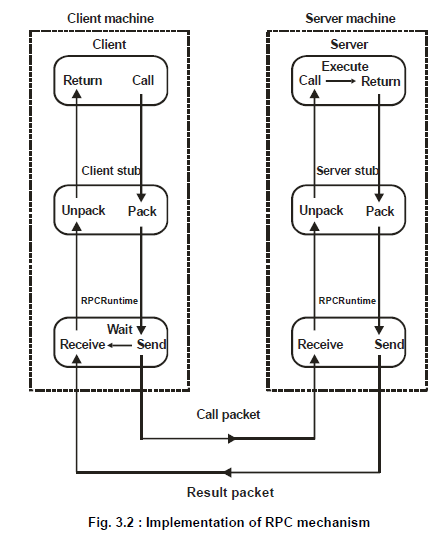
1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's as sends the message to the remote as.
4. The remote as gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local as.
8. The server's as sends the message to the client's as.
9. The client's as gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

**Implementation of RPC**

The implementation of an RPC mechanism is based on the concept of stubs, which provide a perfectly normal (local) procedure call abstraction by concealing from programs the interface to the underlying RPC system. We saw that an RPC involves a client process and a server process. Therefore, to conceal the interface of the underlying RPC system from both the client and server processes, a separate stub procedure is associated with each of the two processes. Moreover, to hide the existence and functional details of the underlying network, an RPC communication package (known as RPCRuntime) is used on both the client and server sides. Thus, implementation of an RPC mechanism usually involves the following five elements of program

1. The client
2. The client stub
3. The RPCRuntime
4. The server stub
5. The server

The interaction between them is shown in Figure 4.2. The client, the client stub, and one instance of RPCRuntime execute on the client machine, while the server, the server stub, and another instance of RPCRuntime execute on the server machine. The job of each of these elements is described below.

****

**Remote Method Invocation (RMI)**

RMI stands for Remote Method Invocation. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

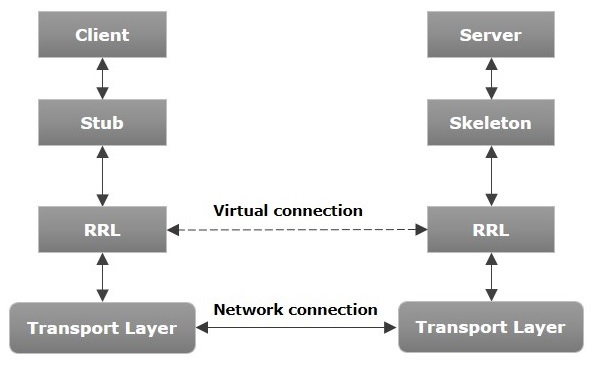
RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package java.rmi.

## Architecture of an RMI Application

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.

The following diagram shows the architecture of an RMI application.

****

Let us now discuss the components of this architecture.

* 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.

## 

## Working of an RMI Application

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.

## 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.

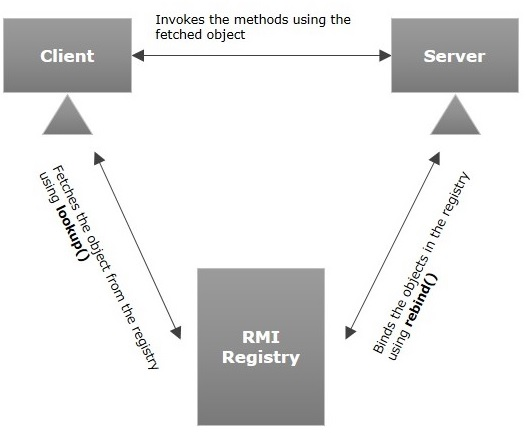
At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as unmarshalling.

## 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).

The following illustration explains the entire process −



## Goals of RMI

Following are the goals of RMI −

* To minimize the complexity of the application.
* To preserve type safety.
* Distributed garbage collection.
* Minimize the difference between working with local and remote objects.

To write an RMI Java application, you would have to follow the steps given below −

* Define the remote interface
* Develop the implementation class (remote object)
* Develop the server program
* Develop the client program
* Compile the application
* Execute the application

## Defining the Remote Interface

A remote interface provides the description of all the methods of a particular remote object. The client communicates with this remote interface.

To create a remote interface −

* Create an interface that extends the predefined interface Remote which belongs to the package.
* Declare all the business methods that can be invoked by the client in this interface.
* Since there is a chance of network issues during remote calls, an exception named RemoteException may occur; throw it.

## Developing the Implementation Class (Remote Object)

We need to implement the remote interface created in the earlier step. (We can write an implementation class separately or we can directly make the server program implement this interface.)

To develop an implementation class −

* Implement the interface created in the previous step.
* Provide implementation to all the abstract methods of the remote interface.

## Developing the Server Program

An RMI server program should implement the remote interface or extend the implementation class. Here, we should create a remote object and bind it to the RMIregistry.

To develop a server program −

* Create a client class from where you want invoke the remote object.
* Create a remote object by instantiating the implementation class as shown below.
* Export the remote object using the method exportObject() of the class named UnicastRemoteObject which belongs to the package java.rmi.server.
* Get the RMI registry using the getRegistry() method of the LocateRegistry class which belongs to the package java.rmi.registry.
* Bind the remote object created to the registry using the bind() method of the class named Registry. To this method, pass a string representing the bind name and the object exported, as parameters.

## Developing the Client Program

Write a client program in it, fetch the remote object and invoke the required method using this object.

To develop a client program −

* Create a client class from where your intended to invoke the remote object.
* Get the RMI registry using the getRegistry() method of the LocateRegistry class which belongs to the package java.rmi.registry.
* Fetch the object from the registry using the method lookup() of the class Registry which belongs to the package java.rmi.registry.  
  To this method, you need to pass a string value representing the bind name as a parameter. This will return you the remote object.
* The lookup() returns an object of type remote, down cast it to the type Hello.
* Finally invoke the required method using the obtained remote object.

## Compiling the Application

To compile the application −

* Compile the Remote interface.
* Compile the implementation class.
* Compile the server program.
* Compile the client program.

**Exercise:**

1. In RMI Architecture which layer Intercepts method calls made by the client/redirects these calls to a remote RMI service?
   1. Stub & Skeleton Layer
   2. Application Layer
   3. Remote Reference Layer
   4. Transport Layer
2. An RMI Server is responsible for \_\_\_\_\_\_\_
   1. Creating an instance of the remote object
   2. Exporting the remote object
   3. Binding the instance of the remote object to the RMI registry
   4. All mentioned above
3. Which program obtains a remote reference to one or more remote objects on a server and then invokes methods on them in an RMI application?
   1. Server
   2. Client
   3. Both A & B
   4. None of the above
4. Which method in naming class specifies a name to the remote object?
   1. bind(string name)
   2. rebind(string name)
   3. Both A & B
   4. None of the above

**Result and Discussion:** .…………………………………………………………………………………………………

…………………………………………………………………………………………………..

**Learning Outcomes:** The student should have the ability to

LO1: Understand the concept of Remote Procedure and Remote Method Invocation

LO 2: Apply RPC, RMI for communication.

**Course Outcomes:** Upon completion of the course students will be able to understand communication using Remote Method Invocation and Remote Procedure Call.

**Conclusion:**……………………………………………………………………………………

…………………………………………………………………………………………………

………………………………………………………………………………………………....

**Viva Questions:**

1. What is the basic principle of RMI architecture ?
2. What are the layers of RMI Architecture ?
3. What is the role of Remote Interface in RMI ?
4. What are the steps involved to make work a RMI program ?
5. What is remote procedure call ?
6. What is Server Stub in remote procedure call?
7. What is marshaling in remote procedure call?
8. What is the sequence of events during remote procedure call?
9. Which transport protocol is used by remote procedure call (RPC)?

**For Faculty Use**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Correction Parameters** | **Formative Assessment [40%]** | **Timely completion of Practical [ 40%]** | **Attendance / Learning Attitude [20%]** |  |
| **Marks Obtained** |  |  |  |