RMI Distributed Applications

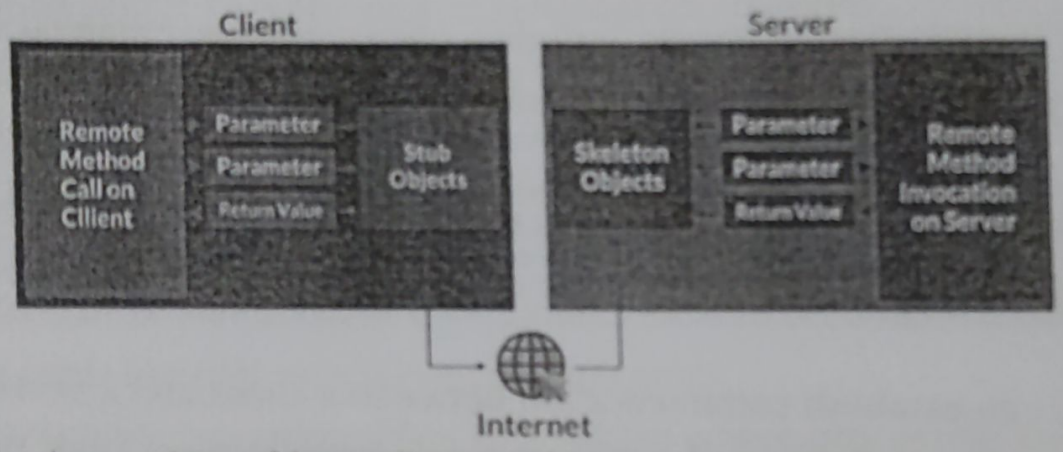
The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects ***stub*** and ***skeleton***.

With RMI, a Java programmer can create a publicly accessible remote server object. This object facilitates seamless client-server communications through simple method calls on the server object. Client programs can communicate directly with the server object and with each other using a URL and HTTP.

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) as also can be depicted from below media as follows

Working of RMI



A high-level overview of how client and server communicate through remote objectsusing RMI is described below.

1. ***Define the Remote Interface:***
   - You define a Java interface that extends the **java.rmi.Remote interface.**
   - Each method in this interface must declare **java.rmi.RemoteException** inits throws clause to handle remote method invocation errors.
   - This interface defines the methods that the client can invoke on theremote object.
2. ***Implement the Remote Object:***
   - You implement the remote interface on the server side. This class will extend **java.rmi.server.UnicastRemoteObject** or java.rmi.server.RemoteObject and implement the methods defined in the remote interface.
   - The server class provides the implementation for the methods declared in the remote interface.
3. ***Create and Start the RMI Registry:***

- The server creates an RMI registry, which acts as a central registry for remote objects. The registry listens for incoming requests on a specific port.
- You can start the RMI registry from the command line using the **rmiregistry** tool provided with the JDK.

4. ***Bind the Remote Object to the Registry:***
   - The server binds the remote object to the RMI registry using a unique name.
   - This makes the remote object accessible to clients by its name.
5. ***Lookup the Remote Object on the Client Side:***
   - The client looks up the remote object in the RMI registry using the naming service **(java.rmi.Naming or java.rmi.registry.LocateRegistry).**
   - The client obtains a reference to the remote object, which it can then use to invoke remote methods.
6. ***Invoke Remote Methods:***
   - The client invokes methods on the remote object reference obtained from the RMI registry.
   - RMI handles the communication details, including parameter organizing, network communication, and error handling, transparently to the client.
7. ***Handle Exceptions:***
   - Both the client and server should handle **java.rmi.RemoteException** and any other application-specific exceptions that may occur during remote method invocation.

By following these steps, you can establish communication between a client and a server using remote objects in Java RMI. This mechanism allows for transparent communication between distributed Java applications, enabling method invocation across different JVMs and physical machines.

Object persistence and serialization

Object persistence and serialization are two related concepts in Java that deal with the storage and retrieval of Java objects in a persistent form, typically to a file or over a network.

Object Persistence:

- Object persistence refers to the ability to store and retrieve objects beyond the life time of the application's execution.
- With object persistence, the state of an object can be saved to a persistent storage medium (such as a database, file system, or cloud storage) and later can be restored, allowing the application to work with the same data across different runs or even different instances of the application.
- Persistence is essential for applications that need to maintain data integrity, share data between multiple users or instances, or store data for long-term use.

Serialization:

- Serialization is the process of converting an object into a stream of bytes, which can be easily stored or transmitted and later reconstructed to create an identical copy of the original object.
- In Java, serialization is achieved by implementing the Serializable interface, which is a marker interface indicating that the class is serializable.
- Serializable objects can be written to an output stream (e.g., a file or network socket) using an **ObjectOutputStream**, and later read from an input stream usingan ObjectInputStream.
- Serialization allows objects to be easily stored to disk, transmitted over a network,or saved to a database, enabling object persistence.

Hence object persistence and serialization provide a powerful mechanism for storing and retrieving Java objects in a persistent form. They are commonly used in various applications, including data storage, caching, messaging systems, and distributed computing. However, it's essential to consider factors like performance, data integrity, and security when designing and implementing object persistence and serialization solutions.

Introduction to Distributed Computing

Distributed computing is the method of making multiple computers work together to solve a common problem. It makes a computer network appear as a powerful single computer that provides large-scale resources to deal with complex challenges.

It is a collection of independent components located on different machines that sharemessages with each other in order to achieve common goals.

Distributed computing refers to a system where processing and data storage is distributed across multiple devices or systems, rather than being handled by a single central device. In a distributed system, each device or system has its own processing capabilities and may also store and manage its own data. These devices or systems work together to perform tasks and share resources, with no single device serving as the central hub.

One example of a distributed computing system is a cloud computing system, where resources such as computing power, storage, and networking are delivered over the Internet and accessed on demand. In this type of system, users can access and use shared resources through a web browser or other client software.

Components

There are several key components of a Distributed Computing System

- **Devices or Systems:** The devices or systems in a distributed system have theirown processing capabilities and may also store and manage their own data.
- **Network:** The network connects the devices or systems in the distributedsystem, allowing them to communicate and exchange data.
- **Resource Management:** Distributed systems often have some type of resource

management system in place to allocate and manage shared resources such as computing power, storage, and networking.

The architecture of a Distributed Computing System is typically a Peer-to-Peer Architecture, where devices or systems can act as both clients and servers and communicate directly with each other.

Characteristics

There are several characteristics that define a Distributed Computing System

- **Multiple Devices or Systems:** Processing and data storage is distributed across multiple devices or systems.
- **Peer-to-Peer Architecture:** Devices or systems in a distributed system can act as both clients and servers, as they can both request and provide services to other devices or systems in the network.
- **Shared Resources:** Resources such as computing power, storage, and networking are shared among the devices or systems in the network.
- **Horizontal Scaling:** Scaling a distributed computing system typically involves adding more devices or systems to the network to increase processing and storage capacity. This can be done through hardware upgrades or by adding additional devices or systems to the network.

Advantages and Disadvantages

Advantages of the Distributed Computing System are:

- **Scalability:** Distributed systems are generally more scalable than centralized systems, as they can easily add new devices or systems to the network to increase processing and storage capacity.
- **Reliability:** Distributed systems are often more reliable than centralized systems, as they can continue to operate even if one device or system fails.
- **Flexibility:** Distributed systems are generally more flexible than centralized systems, as they can be configured and reconfigured more easily to meet changing computing needs.

There are a few *limitations* to Distributed Computing System

- **Complexity:** Distributed systems can be more complex than centralized systems, as they involve multiple devices or systems that need to be coordinated and managed.
- **Security:** It can be more challenging to secure a distributed system, as security measures must be implemented on each device or system to ensure the security of the entire system.
- **Performance:** Distributed systems may not offer the same level of performance as centralized systems, as processing and data storage is distributed across multiple devices

or systems.

Applications

Distributed Computing Systems have a number of applications, including:

- **Cloud Computing:** Cloud Computing systems are a type of distributed computing system that are used to deliver resources such as computing power, storage, and networking over the Internet.
- **Peer-to-Peer Networks:** Peer-to-Peer Networks are a type of distributedcomputing system that is used to share resources such as files and computing power among users.
- **Distributed Architectures:** Many modern computing systems, such as micro services **architectures**, use distributed architectures to distribute processing and data storage across multiple devices or systems.

Challenges of Distributed Systems

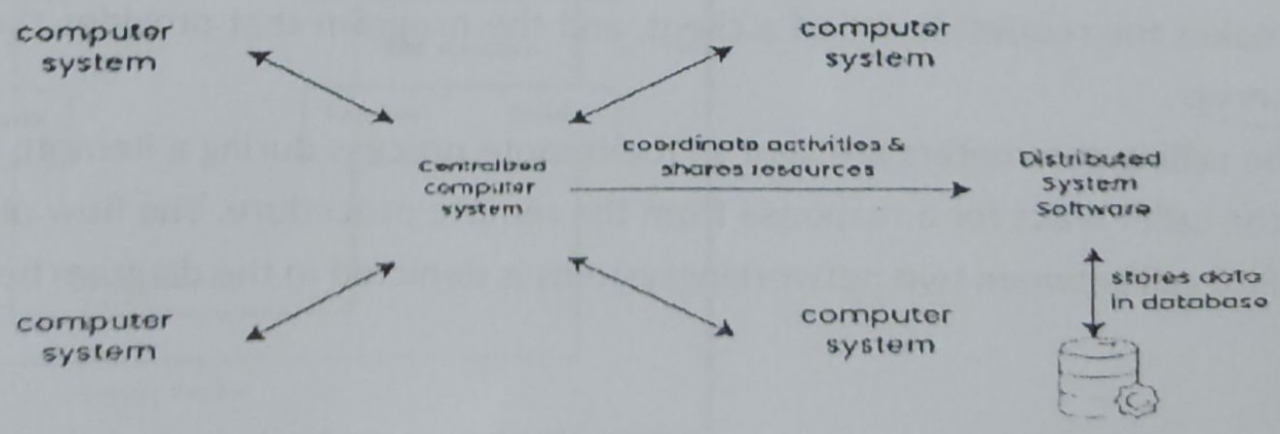
While distributed systems offer many advantages, they also present some challengesthat must be addressed. These challenges include:

- **Network latency:** The communication network in a distributed system can introduce latency, which can affect the performance of the system.
- **Distributed coordination:** Distributed systems require coordination among the nodes, which can be challenging due to the distributed nature of the system.
- **Security:** Distributed systems are more vulnerable to security threats than centralized systems due to the distributed nature of the system.
- **Data consistency:** Maintaining data consistency across multiple nodes in a distributed system can be challenging.

Example of a Distributed System

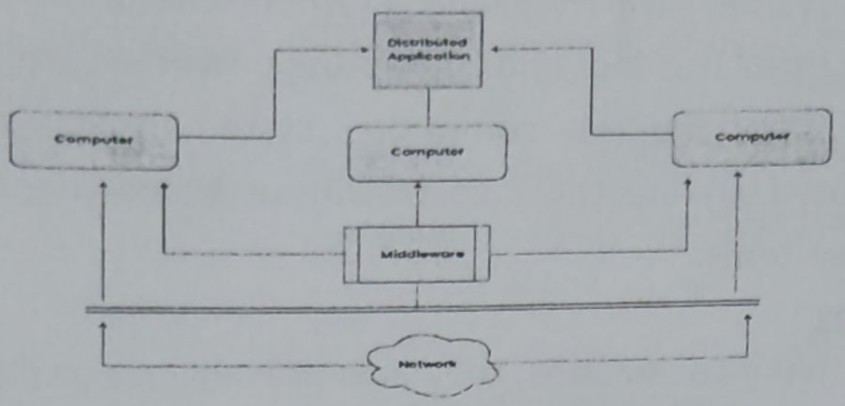
- Any Social Media can have its Centralized Computer Network as its Headquarters and computer systems that can be accessed by any user and using their services will be the Autonomous Systems in the Distributed SystemArchitecture.

Distributed System



- **Distributed System Software:** This Software enables computers to coordinate their activities and to share the resources such as Hardware, Software, Data, etc.**Database:** It is used to store the processed data that are processed by each Node/System of the Distributed systems that are connected to theCentralized network.
- As we can see that each Autonomous System has a common Application thatcan have its

Working of Distributed System

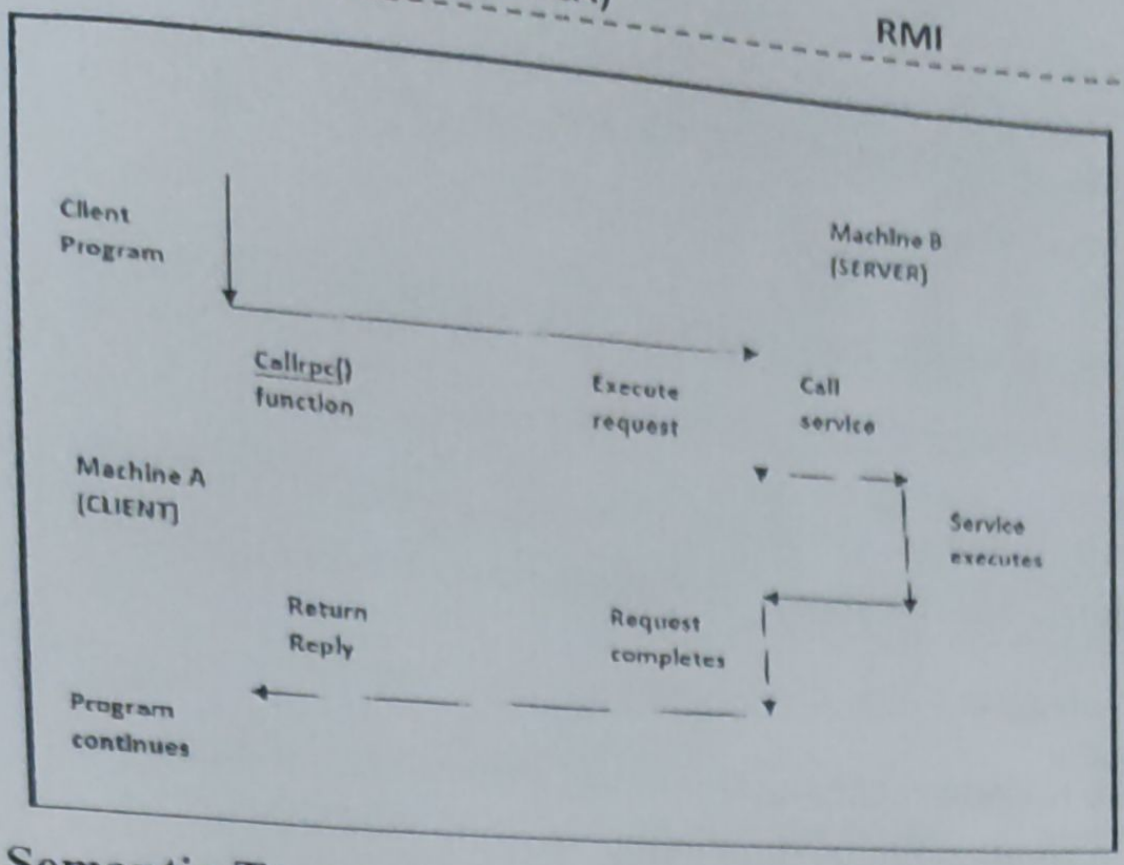


own data that is shared by the Centralized Database System.

- To Transfer the Data to Autonomous Systems, Centralized System should be having a Middleware Service and should be connected to a Network.
- Middleware Services enable some services which are not present in the local systems or centralized system default by acting as an interface between the Centralized System and the local systems. By using components ofMiddleware Services systems communicate and manage data.
- The Data which is been transferred through the database will be divided into segments or modules and shared with Autonomous systems for processing.
- The Data will be processed and then will be transferred to the Centralized system through the network and will be stored in the database.

RPC Implementation Mechanism in DistributedSystem

- RPC is an effective mechanism for building client-server systems that are distributed. RPC enhances the power and ease of programming of the client/server computing concept. It's a protocol that allows one software to seek a service from another program on another computer in a network without having to know about the network. The software that makes the request is called a client, and the program that provides the service is called a server.
- The calling parameters are sent to the remote process during a Remote Procedure Call, and the caller waits for a response from the remote procedure. The flow of activities during an RPC call between two networkingsystems is depicted in the diagram below.



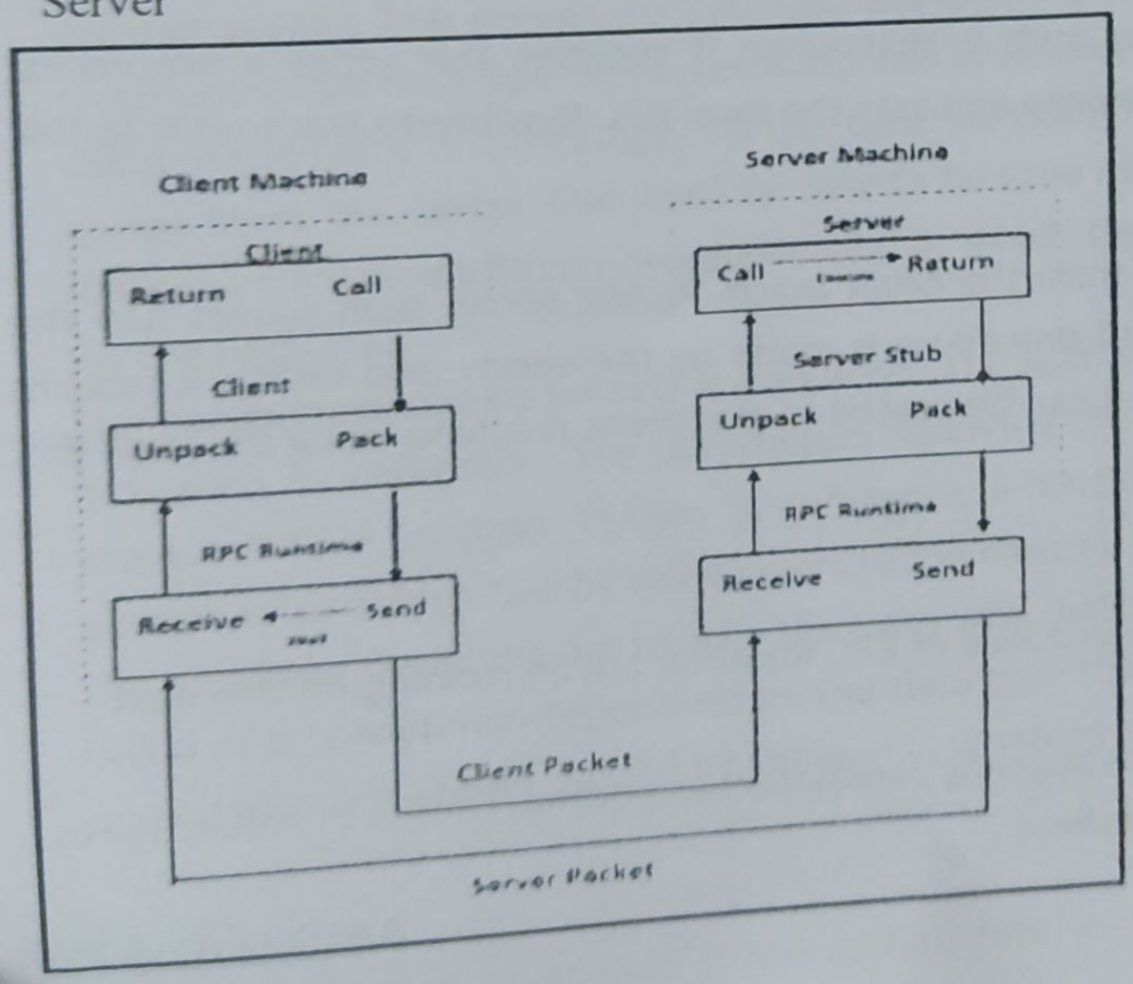
Semantic Transparency:

- **Syntactic transparency:** This implies that there should be a similarity between theremote process and a local procedure.
- **Semantic transparency:** This implies that there should be similarity in the semantics i.e. meaning of a remote process and a local procedure.

Working of RPC:

There are 5 elements used in the working of RPC:

- Client
- Client Stub
- RPC Runtime
- Server Stub
- Server



Client: The client process initiates RPC. The client makes a standard call, which triggers a correlated procedure in the client stub.

Client Stub: Stubs are used by RPC to achieve semantic transparency. The client calls the client stub. Client stub does the following tasks:

The first task performed by client stub is when it receives a request from a client, it packs(marshalls) the parameters and required specifications of remote/target procedure in a message.

The second task performed by the client stub is upon receiving the result values after execution, it unpacks (unmarshalled) those results and sends them to the Client.

RPC Runtime: The RPC runtime is in charge of message transmission between client and server via the network. Retransmission, acknowledgement, routing, and encryption are all tasks performed by it. On the client-side, it receives the result valuesin a message from the server-side, and then it further sends it to the client stub whereas, on the server-side, RPC Runtime got the same message from the server stubwhen then it forwards to the client machine. It also accepts and forwards client machine call request messages to the server stub.

Server Stub: Server stub does the following tasks:

The first task performed by server stub is that it unpacks(unmarshalled) the call requestmessage which is received from the local RPC Runtime and makes a regular call to invoke the required procedure in the server.

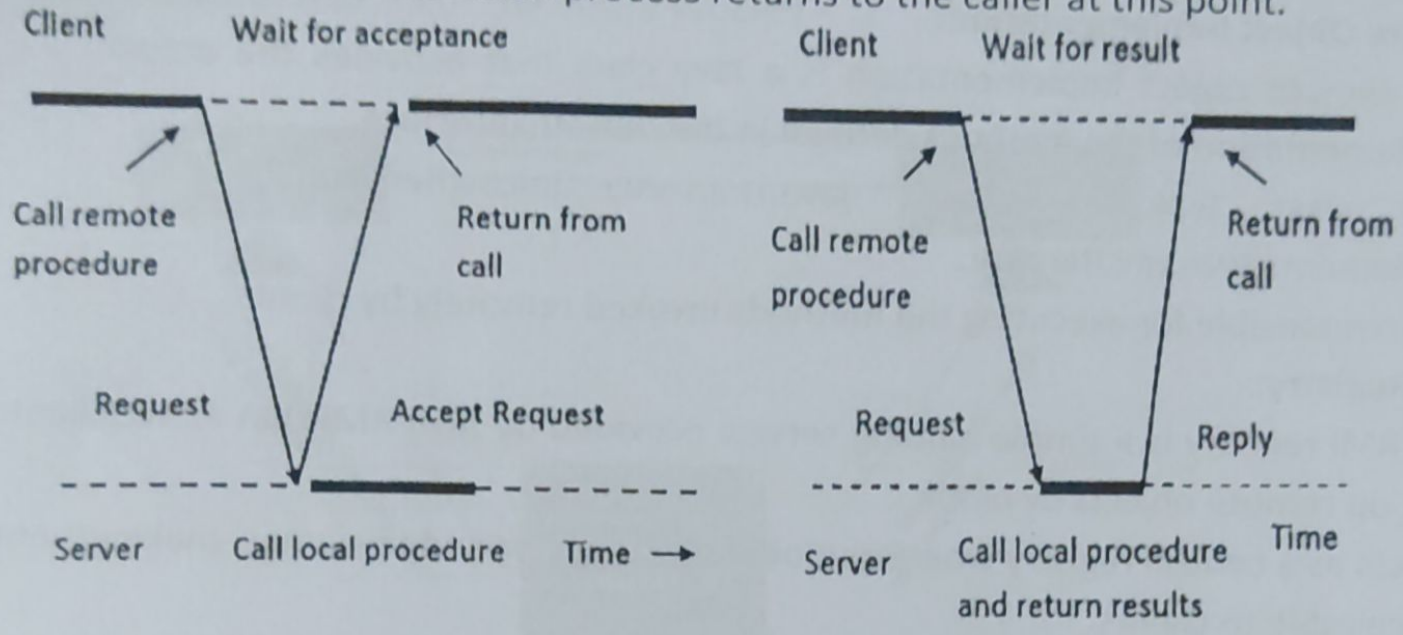
The second task performed by server stub is that when it receives the server's procedure execution result, it packs it into a message and asks the local RPC Runtimeto transmit it to the client stub where it is unpacked.

Server: After receiving a call request from the client machine, the server stub passes it to the server. The execution of the required procedure is made by the server and finally, it returns the result to the server stub so that it can be passed to the client machine using the local RPC Runtime.

RPC process:

- The client, the client stub, and one instance of RPC Runtime are all running on theclient machine.
- A client initiates a client stub process by giving parameters as normal. The client stubacquires storage in the address space of the client.

- At this point, the user can access RPC by using a normal Local Procedural Call. The RPC runtime is in charge of message transmission between client and server via the network. Retransmission, acknowledgment, routing, and encryption are all tasks performed by it.
- On the server-side, values are returned to the server stub, after the completion of server operation, which then packs (which is also known as marshaling) the return values into a message. The transport layer receives a message from the server stub.
- The resulting message is transmitted by the transport layer to the client transport layer, which then sends a message back to the client stub.
- The client stub unpacks (which is also known as unmarshalling) the return arguments in the resulting packet, and the execution process returns to the caller at this point.



- When the client process requests by calling a local procedure then the procedure will pass the arguments/parameters in request format so that they can be sent in a message to the remote server. The remote server then will execute the local procedure call (based on the request arrived from the client machine) and after execution finally returns a response to the client in the form of a message. Till this time the client is blocked but as soon as the response comes from the server side itwill be able to find the result from the message. In some cases, RPCs can be executed asynchronously also in which the client will not be blocked in waiting for the response.
- The parameters can be passed in two ways. The first is to pass by value, whereas the second is to pass by reference. The parameters receiving the address should be pointers when we provide it to a function. In Pass by reference, a function is called using pointers to pass the address of variables. Call by value refers to the method ofsending variables' actual values.
- The language designers are usually the ones who decide which parameter passing method to utilize. It is sometimes dependent on the data type that is being provided. Integers and other scalar types are always passed by value in C, whereas arrays arealways passed by reference.

RMI Architecture

The Remote Method Invocation (RMI) architecture in Java facilitates communication between

Java objects residing in different Java Virtual Machines (JVMs) over a network. It allows objects to invoke methods on remote objects as if they were localobjects. The RMI architecture typically involves several components and follows a client-server model:

1. **Remote Interface:**
   - The remote interface is a Java interface that defines the methods that can beinvoked remotely by clients. It extends the java.rmi.Remote interface.
   - Each method in the remote interface must declare java.rmi.RemoteException inits throws clause to handle remote method invocation errors.
2. **Remote Object Implementation:**
   - The remote object implementation is a Java class that provides the actual implementation of the methods defined in the remote interface.
   - This class typically extends java.rmi.server.UnicastRemoteObject or implements java.rmi.Remote.
   - It is responsible for executing the methods invoked remotely by clients.
3. **RMI Registry:**
   - The RMI registry is a simple naming service provided by Java RMI that allowsclients to look up remote objects by name.
   - It acts as a central registry where remote objects are bound to names, makingthem accessible to clients.
   - The server creates an RMI registry, and clients use it to locate remote objects.
4. **Client:**
   - The client is the application or component that initiates communication withremote objects.
   - It obtains a reference to the remote object from the RMI registry using the namingservice provided by java.rmi.Naming or java.rmi.registry.LocateRegistry.
   - Once it has the reference, the client can invoke methods on the remote object asif it were a local object.
5. **Marshalling and Unmarshalling:**
   - RMI handles the marshalling (serialization) and unmarshalling (deserialization) of method parameters and return values transparently to the client and server.
   - Method parameters and return values are automatically serialized into a stream of bytes before being transmitted over the network and deserialized back into objects at the receiving end.
6. **Network Communication:**
   - RMI uses TCP/IP as the underlying network protocol for communication betweenthe client and server JVMs.
   - It establishes a connection between the client and server JVMs, allowing them toexchange

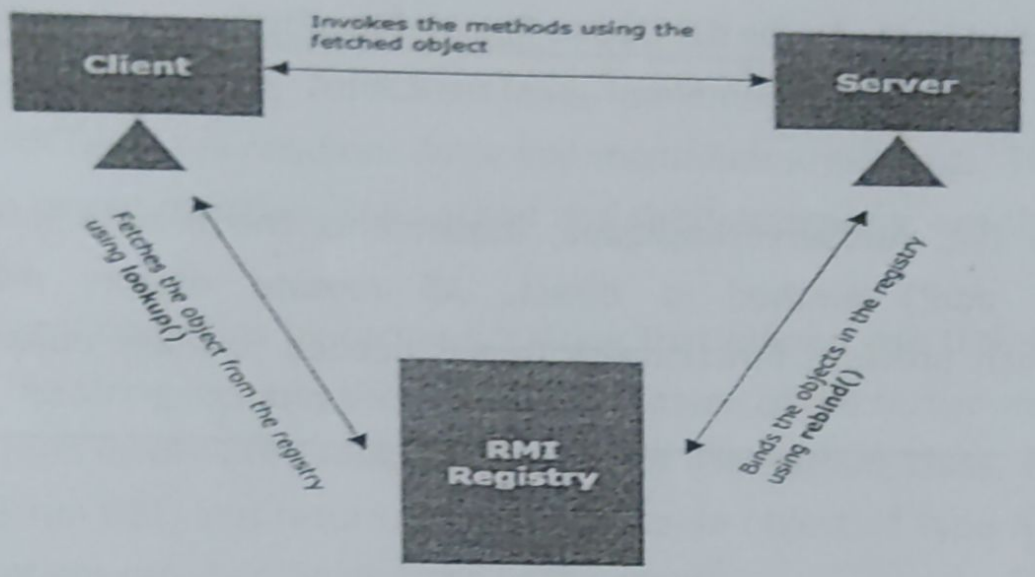
method invocations and data.

Importance of 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. It allows remote clients to get a reference to these objects.

To invoke a remote object, the client needs a reference of that object. At that time, theclient 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.

Callback Implementation in RMI

An RMI callback occurs when the client of one service passes an object ,that is theproxy, for another service. The recipient can then call methods in the object it received, and be calling back (hence the name) to where it came from

With RMI, a Java programmer can create a publicly accessible remote server object. This object facilitates seamless client-server communications through simple method calls on the server object. Client programs can communicate directly with the server object and with each other using a URL and HTTP.

Developing Simple RMI application

In any RMI application, the server receives a request from a client, processes it, and returns a result. In this example, the request specifies two numbers. The server adds these together and

returns the sum.

Step One: Enter and Compile the Source Code

This application uses four source files.

The first file, **AddServerIntf.java**, defines the remote interface that is provided by the server. It contains one method that accepts two double arguments and returns their sum. All remote interfaces must extend the Remote interface, which is part of java.rmi. Remote defines no members. Its purpose is simply to indicate that an interface uses remote methods. All remote methods can throw a RemoteException.

```
import java.rmi.*;

public interface AddServerIntf extends Remote

{ double add (double d1, double d2) throws RemoteException;

}
```

The second source file, **AddServerImpl.java**, implements the remote interface. The implementation of the add() method is direct. All remote objects must extend UnicastRemoteObject, which provides functionality that is needed to make objects available from remote machines.

```
import java.rmi.*;
import java.rmi.server.*;
public class AddServerImpl extends UnicastRemoteObject implements AddServerIntf
{
public AddServerImpl() throws RemoteException { }

        public double add (double d1, double d2) throws RemoteException
                {return d1 + d2;
                }
        }
```

The third source file, **AddServer.java**, contains the main program for the server machine. Its primary function is to update the RMI registry on that machine. This is done by using the **rebind()** method of the ***Naming*** class (found in java.rmi). That method associates a name with an object reference. The first argument to the rebind()method is a string that names the server as "AddServer". Its second argument is a reference to an instance of AddServerImpl.

```
import java.net.*;
import java.rmi.*;
public class AddServer
{
        public static void main(String args[])
        {
        try {
                        AddServerImpl addServerImpl= new AddServerImpl();
                        Naming.rebind("AddServer", addServerImpl);
                        System.out.println("Server is running.....");
                }
         catch (Exception e)
                {System.out.println("Exception:"+e);
                }
        }
}
```

The fourth source file, **AddClient.java**, implements the client side of this distributed application. AddClient.java requires three command-line arguments. The first is the IP address or name of the server machine. The second and third arguments are the two numbers that are to be summed.

The application begins by forming a string that follows the ***URL*** syntax. This URLuses the ***rmi protocol***. The string includes the IP address or name of the server and thestring "AddServer". The program then invokes the **lookup()** method of the Naming class. This method accepts one argument, the rmi URL, and returns a reference to an object of type AddServerIntf. All remote method invocations can then be directed to this object.

The program continues by displaying its arguments and then invokes the remote **add()** method. The sum is returned from this method and is then printed.

```
import java.rmi.*;
public class AddClient
{
public static void main(String args[])
{

    try {
          String addServerURL= "rmi://" + args [0]+"/AddServer";
          AddServerIntf addServerIntf = (AddServerIntf) Naming.lookup (addServerURL);
          System.out.println("The first number is: "+ args[1]);
          double d1 = Double.valueOf(args[1]).doubleValue();
          System.out.println("The second number is: "+ args [2]);
          double d2 = Double.valueOf(args[2]).doubleValue();
          System.out.println("The sum is:"+ addServerIntf.add(d1, d2));
          }
    catch (Exception e)
     {    System.out.println("Exception: " + e); }

}

}
```

Compile all the four source code file using **javac**.

Step Two: Generate a Stub

Next we need to generate the necessary stub. In the context of RMI, ***a stub is a Java object that resides on the client machine.*** Its function is to present the same interfacesas the remote server.

Remote method calls initiated by the client are actually directedto the stub. The stub works with the other parts of the RMI system to formulate a request that is sent to the remote machine.

A remote method may accept arguments that are simple types or objects. All of this information must be sent to the remote machine. That is, an object passed as an argument to a remote method call must de serialized and sent to the remote machine.

If a response must be returned to the client, the process works in reverse. Theserialization and deserialization facilities are also used if objects are returned to a client.

To generate a stub, we use a tool called the RMI compiler, which is invoked fromthe command line, as shown here:

rmic AddServerImpl

This command generates the file ***AddServerImpl_Stub.class***. When using rmic,be sure that CLASSPATH is set to include the current directory.

Step Three: Install Files on the Client and Server Machines

Copy AddClient.class, AddServerImpl_Stub.class, and AddServerIntf.class to a directory Stub.class, and AddServer.class to a directory on the server machine.

RMI has techniques for dynamic class loading, but they are not used by the example at hand. Instead, all of the files that are used by the client and server applications must be installed manually on those machines.

Step Four: Start the RMI Registry on the Server Machine

Java SE 6 provides a program called **rmiregistry**, which executes on the server machine. It maps names to object references. First, check that the CLASSPATH environment variable includes the directory in which your files are located. Then, start the RMI Registry from the command line, as shown here:

start rmiregistry

When this command returns, a new window has been created. This window must beopen until experimenting with the RMI is done.

Step Five: Start the Server

The server code is started from the command line, as shown here:

java AddServer

Recall that the AddServer code instantiates AddServerImpl and registers that object with the name "AddServer".

Step Six: Start the Client

The Add Client software requires three arguments: the name or IP address of the servermachine and the two numbers that are to be summed together. We may invoke it fromthe command line

by using one of the two formats shown here:

java AddClient **server1** 8 9

java AddClient **11.12.13.14** 8 9

In the first line, the name of the server is provided. The second line uses its IP address (11.12.13.14)

We can try this example without actually having a remote server. To do so, simply install all of the programs on the same machine, **start rmiregistry, start AddServer**,and then **execute AddClient** using this command line:

java AddClient 127.0.0.1 8 9

Here, the address 127.0.0.1 is the "loop back" address for the local machine. Using this address allows you to exercise the entire RMI mechanism without actually havingto install the server on a remote computer.

Sample output from this program is shown here:

The first number is: 8 The second number is: 9The sum is: 17.0