RMI

Remote Method Invocation

Overview

- We have already know What is n – tier architecture of j2ee.
- Now we are going to discuss what is distributed application?
- It is an application which is divided into more than one machine and machines are attached using network.
- So, whole application is distributed within two or more machines.
- In this distribution, one part of the whole application has authority to get request from other part of application,

Overview

- Which is known as Application Server.
- Whereas the other part of application has authority to make a request to the server, which is known as Client Application.
- Now a day's most of the development and deployment environment is in network. So, distributed application becomes so popular.
- For distributed application client / server system is implemented.

Overview

- So, here process of distributed application is distributed across multiple networked computers.
- We know that in client/server system client makes a request to the server and the server fulfills its request and provides appropriate response to the client.
- There are three components of client server system:
  - Information Layer(Presentation Layer)
  - Information Processing Layer
  - Information Storage Layer

Overview

- Presentation Layer is implemented by client system sometimes it is also known as web browser.
- But the information processing layer is implemented either by client or server or server support system and it depends upon the architecture of the application.
- The information storage layer is implemented on web server or FTP sever or J2EE server.

Overview

- According to the behavior of the application information processing layer may be at client side or it may be at server side.
- We have already discussed concept of thin client and thick client.
- In thin client processing layer resides at server side.
- And in the thick client processing layer resides at client side.

Overview

- In distributed application, following types of method invocation are used.
  - Distributed Computing Environment (DCE)
  - Distributed Components Object Model (DCOM)
  - Common Object Request Broker Architecture (CORBA)
  - Java's Remote Method Invocation (RMI)

DCE

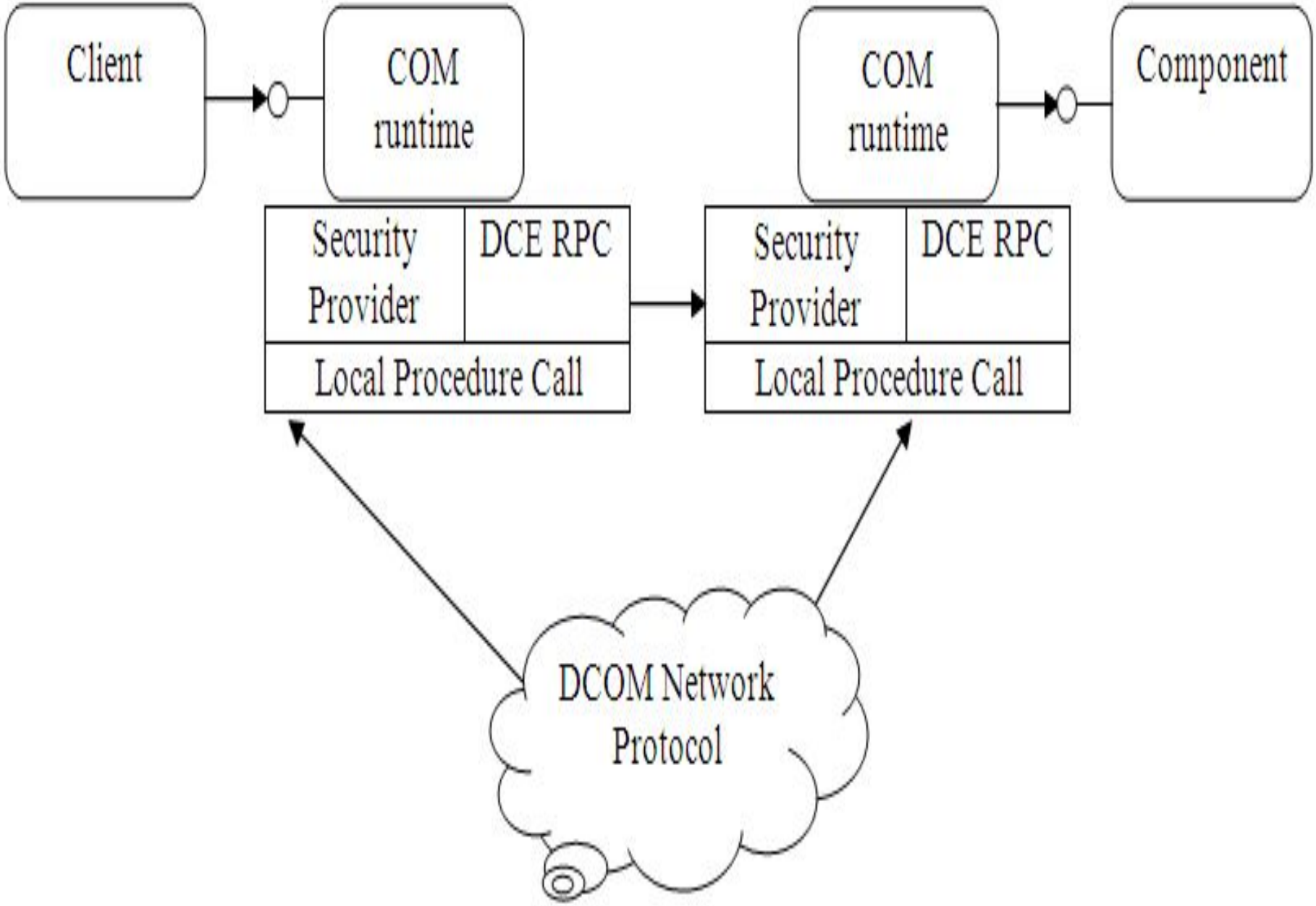
- Distributed Computing Environment is an industry standard for Open System Foundation (OSF).
- It is vendor – natural set of distributed computing.
- It provides security and control for accessing data.
- In distributed environment all the components that provides individual functionality is calling DCE cell.

- For ex, a college system is divided into different departments like chemistry, biology, maths, computer science, admin etc.
- In each DCE cell many services and technologies are used which are described as follows:
  - Directory Services
  - Distributed File System (DFS)
  - Distributed Time Services (DTS)
  - Security Services
  - Remote Procedure Call (RPC)
  - DCE Threads

DCOM

- DCOM is a proprietary of Microsoft Corporation.
- DCOM is a solution for OLE (Object Linking and Embedding) and its design is different from DCE.
- DCOM supports object oriented platform independent components.
- Whereas DCE supports Procedure oriented components.
- DCOM is made up of COM components.

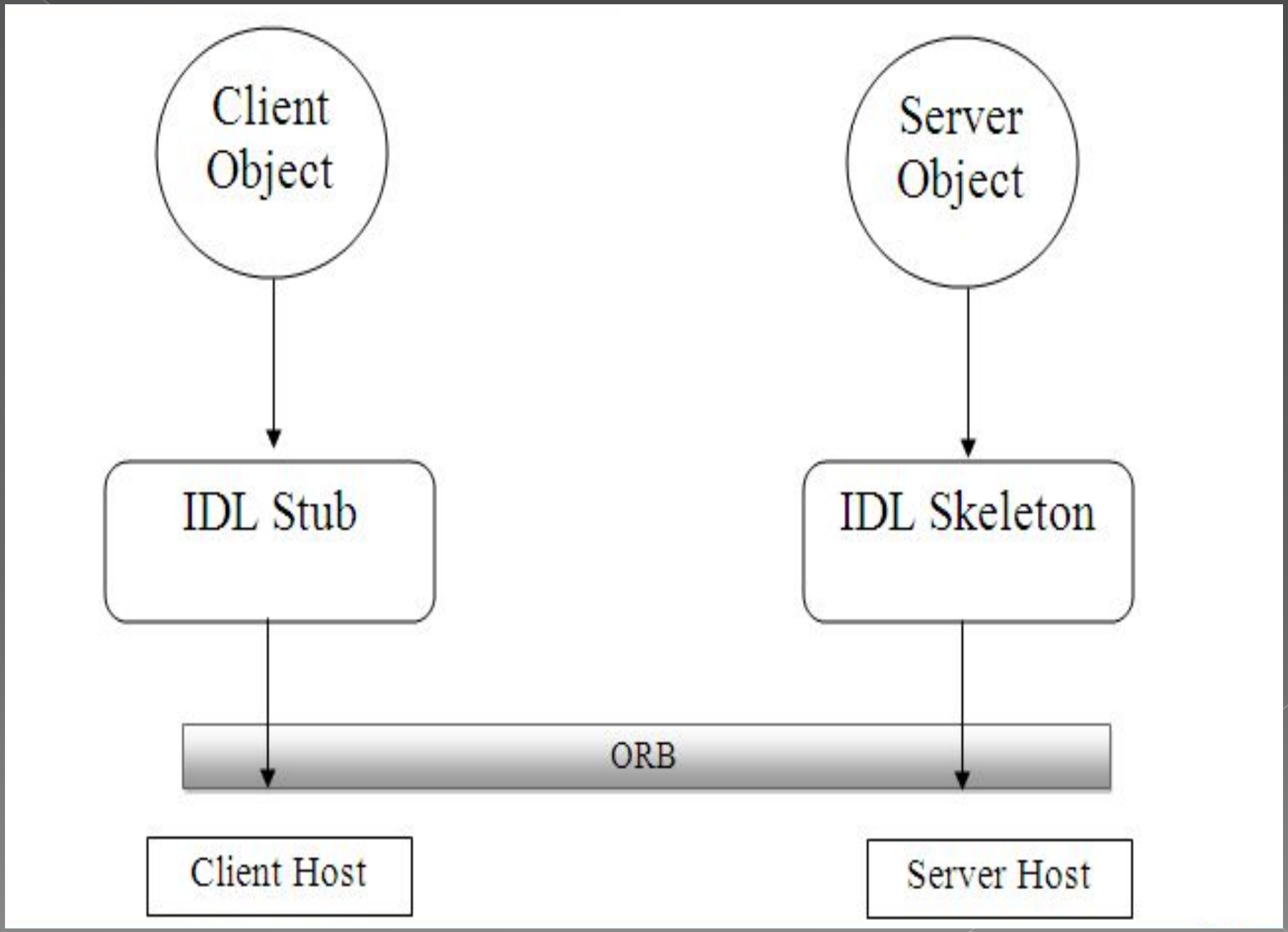
- DCOM allows COM object on one computer to access methods of another COM object on other computers.
- COM objects can be created with a variety of Object – oriented languages, programming languages, such as C++ provide programming mechanisms that simplify the implementation of COM objects.
- According to figure, DCOM is distributed COM over multiple computers. Local object processes the function calls using COM runtime in term of Object RPC.


Client
COM runtime
COM runtime
Component
Security Provider
DCE RPC
Security Provider
DCE RPC
Local Procedure Call
Local Procedure Call
DCOM Network Protocol

CORBA

- CORBA is a most popular architecture for building distributed application.
- Using CORBA we can develop object oriented distributed applications.
- Like other architecture architecture CORBA is not proprietary architecture.
- It is an open standard and objects are accessed with ORBs (Object Request Brokers).
- Client interface of ORB is known as IDL (Interface Definition Language)

- Stub and server interface is known as IDL Skeleton.
- Method of invocation in CORBA is shown in figure.


Client Object
Server Object
IDL Stub
IDL Skeleton
ORB
Client Host
Server Host

- Advantages of CORBA are that due to Open Standard it works on all platforms.
- Clients and Servers can be distributed and language independent.
- So, we have to provide only IDL interface to object.

RMI

- RMI is for java's distributed object model.
- Now, we have a question that why we need RMI though we have DCE, DCOM, and CORBA?
- And here is the answer: Remote Procedure Call of DCE is not suitable with Java's Object Oriented Distributed Application (OODA).
- DCOM has good performance with windows because it is a proprietary of Microsoft so, DCOM does not support java object fully.

- CORBA is excellent but java programmer will have to learn IDL (Interface Definition Language)to develop Java Distributed Application (JDA).
- All these methods have their own advantages and disadvantages in terms of complexity.
- Performance and cross – platform compatibility.
- But RMI is purely Java – specific which provides java to java communications only and as a result, RMI is much simpler than CORBA.

Introduction to RMI

- As we discussed before DCE, DCOM and CORBA are not compatible for J2EE environment.
- So, RMI is invented for J2EE distributed application.
- J2EE provides the distributed programming environment for the users.
- It is having number of APIs to develop Enterprise Application.
- RMI is the one of the API for developing the distributed Application.

- Distributed application contains the distributed objects over the network.
- It communicates with each other by using Transport protocol or basically using TCP/IP protocol.
- Client sends request to the server to access remote objects and server sends the response related to client request.
- This kind of client – server computing is possible with java by using RMI.
- Client can access the remote objects by invoking the methods.

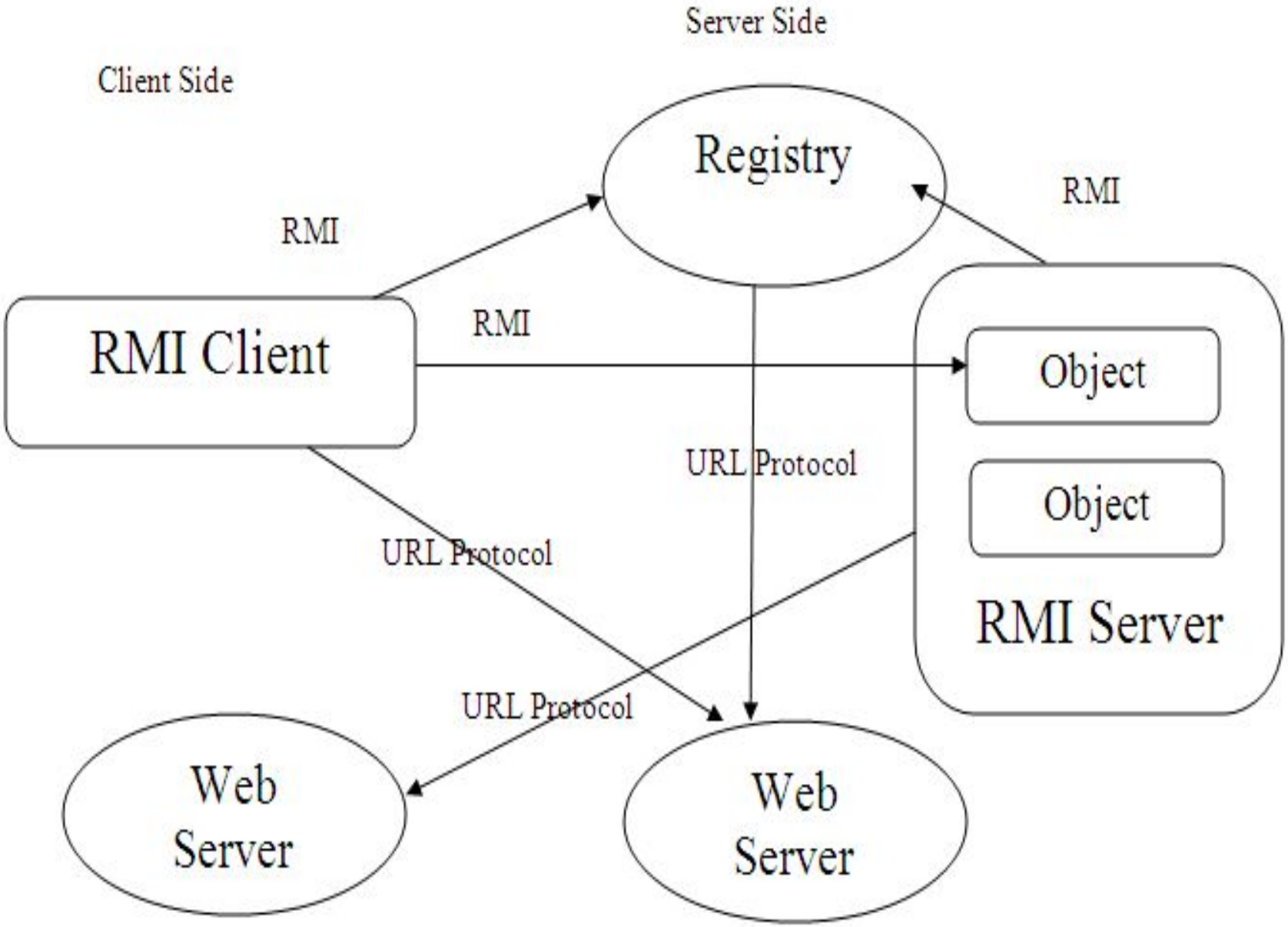
- Definition of RMI:
  - RMI is the technology which allows the client to Remote Object communication or object to object communication between different JVM.
  - In short, RMI provides remote communication between programs written in the java programming language.

An overview of RMI Application

- RMI applications basically divide in two programs client and server.
- A server program creates some remote objects and makes the references to them and waits for the client request to invoke methods on the remote objects.
- A client application gets a remote reference to one or more Remote objects from the server and invokes the methods on them.
- RMI provides the mechanism by which the server and the client communication and pass information back and forth.

- Such application is to say a distributed object application (DOA).
- Distributed Object Application has to include following three things:
  - Locate remote objects: an application can use various mechanisms to obtain reference to remote objects.
  - For example, an application can register its remote objects with the RMI registry.
  - Alternatively, an application can pass and return remote object references as part of other remote invocations.
  - Communicate with remote objects: all communication between remote objects is handled by RMI. Remote communication looks similar to regular java method invocations.

- Load class definitions for objects that are passed around:
  - Because RMI enables objects to be passed back and forth, it provides mechanisms for loading an object's class definitions as well as for transmitting an object's data.


Server Side
Client Side
Registry
RMI
RMI
RMI Client
RMI
Object
URL Protocol
Object
URL Protocol
RMI Server
URL Protocol
Web Server
Web Server

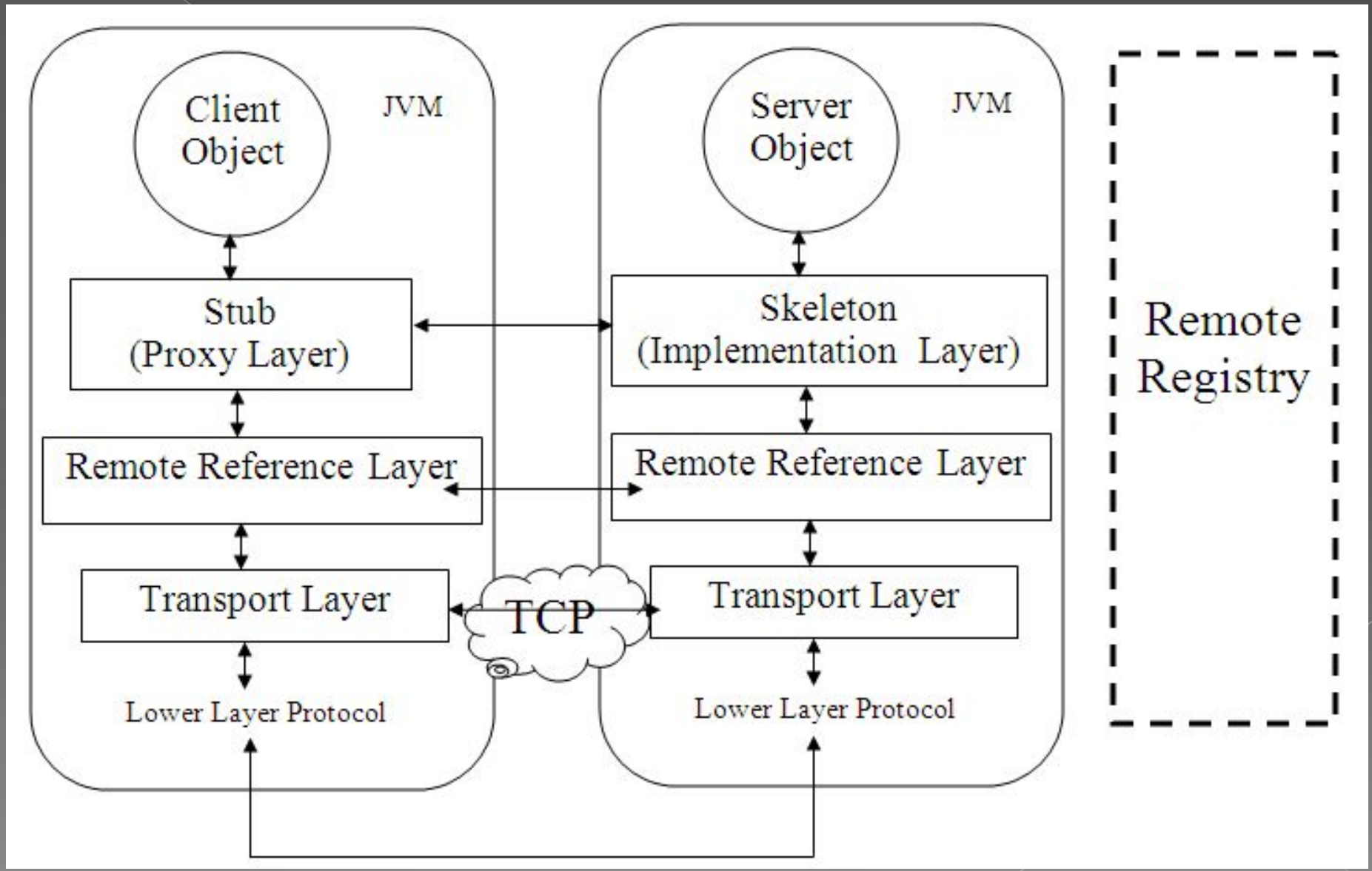
RMI Architecture

- RMI architecture defines the how the client request to the server for the remote objects and how the server processes for that request.
- It also defines the implementation of remote class on a remote JVM.
- RMI architecture basically divides in 4 layers:

- Application layer
- Proxy layer (Stub and Skeleton)
- Remote reference layer
- Transport layer
- Each layer can perform specific functions, like establish the connection, assemble and disassemble the parameters, transmitting the object etc.
- Above RMI layers are independent from each other and they can be used based on the client request for the service as shown in figure below.

- The garbage collection is one of the advantages of the java language.
- RMI also supports garbage collection.
- The registry simply keeps track of the addresses of remote objects that are being exported by the applications.
- All the distributed systems use the registry to keep track of the names of the remote objects.

RMI architecture



Application Layer

- Application layer consists of the client side java program and the server side java program with the remote methods.
- Here, the high level calls are made in order to access and export remote objects.
- The client can access the remote method through an interface that extends java.rmi.Remote.
- When we want to define a set of methods that will be remotely called they must be declared in one or more interface that should extend java.rmi.Remote.

- Once the methods described in the remote interface have been implemented, the object must be exported using UnicastRemoteObject class of the java.rmi.server package.
- The application will register itself with registry and it is used by client to obtain reference of the objects.
- On the client side a client simply request a remote object from either registry or remote object whose reference has already been obtained.

- After getting reference the proxy layer has to perform the important role of communication between the client and server in out application, the RemoteMessageClient and RemoteMessageServer from application layer.

Proxy Layer

- The stub / skeleton layer is responsible for listening to the remote method calls by the client redirecting them to the server.
- This layer consists of a stub and a skeleton.
- The stub and skeleton are created using the RMI compiler (RMIC).
- These are simply class files that represent the client and server side remote object.
- A stub is a client side proxy representing the remote objects.

- Stub communicates the method invocations to the remote object through a skeleton that is implemented on the server.
- The skeleton is a server side proxy that communicates with the stub.
- It makes a call to the remote object.
- Therefore, it is also known as proxy layer.
- It is also interface between application layer and all RMI layers.

- Use of rmic is:
- Rmic <options> <classname>

Option	Description
-keep or –keepgenerated	Retains the geenrated .java source files for the stub, skeleton and / or tie classes and writes them to the same directory as the .class files.
-v1.1	To create stub/skeleton for jdk 1.1 stub protocol version.
-v1.2	To create stubs for jdk 1.2 stub protocol version only.
-g	To generate debugging information.
-depend	Recompile out of date files recursively.
-nowarn	Generates no warnings
-verbose	Output messages about what compiler is doing.

Options	Description
-classpath <path>	Specify where to find input source and class files.
-d <directory>	Specify where to place generated class file.
-j <runtime flag>	Pass argument to the java interpreter.

Remote Reference Layer

- Remote reference layer is interface between the proxy layer and transport layer, it handles actual communication protocols.
- The remote reference layer (RRL) interprets the references made by the client to the remote object on the server.
- The RRL on the client – side receives the request for the methods from the stub.
- The request is then transferred to the RRL on the server – side.

- When transmitting the parameter or objects through the network it should be in the form of a stream.
- The JVM works with the Java Byte Codes.
- It can get the stream – oriented data from the transport layer and give it to the proxy layer and vise versa.
- RRL is used to:
  - Handle the replicated objects. Once the feature is incorporated into the RMI system, the replicated objects will allow simple dispatch to other programs.
  - It is responsible for establishing persistence and strategies for recovery of lost connections.

Transport Layer

- Transport layer manages the connection between client remote reference layer and remote server machine.
- It receives a request from the client – side RRL and establishes a connection with the server through server – side RRL.
- The transport layer has following responsibilities:
  - It is responsible for handling the actual machine to machine communication; the default communication will take place through a standard TCP/IP.

- It creates a stream that is accessed by the remote reference layer to send and receive data to and from other machine.
- It sets up the connections to remote machines.
- It manages the connections.
- It monitors the connections to make sure that they (remote machines) are live.
- It listens for connections from the machines.

STUB AND SKELETON

- The stub class:
- The stub is a client side proxy of the remote object. It has three primary responsibilities.
  - It presents the same remote interfaces as the object of server (Ex. RemoteMessageServer). Therefore from the perspective of the client (Ex. RemoteMessageClient), the stub is equivalent to the remote object.
  - It works with JVM and RMI system on client machine to serialize any arguments to a remote method call and sends this information to server machine.
  - The stub receives any result from the remote method and returns it to the client.

STUB AND SKELETON

- The skeleton class:
- The skeleton is a server side proxy of the remote object. It has three primary responsibilities.
  - It receives the remote method call and any associated arguments. It works with the JVM and RMI system on the server machine to deserialize any arguments for the remote method call.
  - It invokes the appropriate method in the server (Ex. RemoteMessageServer), using arguments.
  - It receives any return value from this method call and works with the JVM and RMI system on server machine to serialize this return value and sends the information back to client (Ex. RemoteMessageClient).

Developing an RMI Application

- To develop an RMI application, we need to create the following java programs:
  - Remote Interface
  - Remote Object class
  - Server program
  - Client program
- To develop the following java files we need to use java.rmi.* (package) and java.rmi.server package (API).
- Let us take brief introduction of java.rmi package and java.rmi.server package.

java.rmi package

- Java.rmi package is having the following classes and interfaces which is used to develop RMI application with java.
- Remote Interface:
  - It is used to create Remote interface program. We need to extend the Remote interface in remote interface java program.
- Naming class:
  - It is used to identify the remote objects. It also provides host name and IP address of machine which is running on the server to the client. Client can get all these information by using following methods.

Methods of Naming class

- Lookup():
  - It is used to identify the remote object or we can say it returns the name of remote object.
  - It is having a one String parameter URL, which is used to look up the Remote Object.
  - For example: naming.lookup("rmi://hostname/objectname");
  - It should throw the RemoteException and MalformedURLException.
- Bind():
  - It is used to bind the specified remote name to the remote object. It's having two parameters with it first name on the object and second object it self.

Methods of Naming class

- For example: naming.bind("hello client",greeting);

- Rebind():
  - It is used to rebind the name with RemoteObject. It's having two parameters with it first is the string message and second is the new RemoteObject name.
  - For example: naming.rebind("hello",msg);
- Unbind():
  - It is opposite to the bind method. It destroys the binding object with remote object. It is having one string parameter with it.

Methods of Naming class

- For example: naming.unbind("hello");

- list():
  - It returns the string array with the name of objects bound in the registry. It passes one parameter with it.
  - For example: naming.list(URL);

RMISecurityManager class

- It is derived from SecurityManager class of java.lang package.
- Basic form of it is
- public class RMISecurityManager extends SecurityManager.
- It is used to download the code in RMI application.
- Without SecurityManager RMI class loader can not download any class from remotejost.
- To add RMISecurityManager in your application you can use following code:

- System.setSecurityManager(new RMISecurityManager());

- RemoteException:
  - RemoteException is used to handle the runtime errors occurred during the Remote method are used in the application.
  - The interface which inherits the Remote interface of java.rmi package must throw RemoteException.

Java.rmi.server package

- Java.rmi.server package is having the following classes and interfaces which is used to develop RMI application with java.
- RemoteObject class:
  - Remote object class is used to create the RemoteObject which is located on the server but works as client object.
- UnicastRemoteObject:
  - It works same as the RemotObject class because this is a concrte subclass of RemoteServer that implements point – to – point remote references over TCP/ IP networks.