Unit 8: RMI and CORBA

Prepared by: Aakash Raj Shakya

Remote Method Invocation(RMI)

1. **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.
2. RMI is used to build distributed applications; it provides remote communication between Java programs.
3. It is provided in the package **java.rmi.**
4. RMI is divided into two parts:
   a. Client Program: Make request for remote objects on server and invoke method on them.
   b. Server Program: Creates some remote objects.

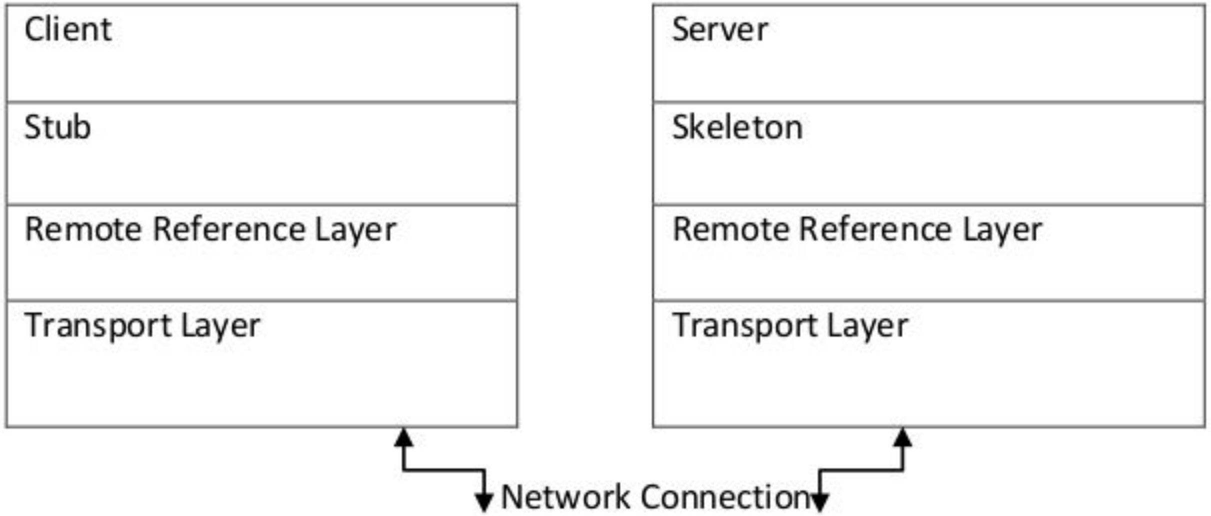
RMI Architecture

The interface that the client and server objects use to interact with each other is provided through **stubs/skeleton, remote reference layer, and transport layers**.

Stubs and **skeletons** are Java objects that act as proxies to the client and server, respectively.

All the network-related code is placed in the stub and skeleton, so that the client and server will not have to deal with the network and sockets in their code.

The remote reference layer handles the creation of and management of remote objects. The transport layer is the protocol that sends remote object requests across the network.


Client
Stub
Remote Reference Layer
Transport Layer
Server
Skeleton
Remote Reference Layer
Transport Layer
Network Connection

Working of RMI application

1. When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
2. 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.
3. 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.
4. The result is passed all the way back to the client.

Adder.java

```java
import java.rmi.*;

public interface Adder extends Remote {

   // Declaring the method prototype
   public int add(int firstNumber, int secondNumber) throws RemoteException ;
}
```

AdderImplements.java

```java
import java.rmi.*;
import java.rmi.server.*;

public class AdderImplements extends UnicastRemoteObject implements Adder {

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

    // Implementation of the adder interface
    @Override
    public int add(int firstNumber, int secondNumber) {
        return (firstNumber + secondNumber);
    }
}
```

Server.java

```
import java.rmi.*;
import java.rmi.registry.*;

public class Server {

   public static void main(String args[]) {
       try {
           // Create an object of the interface implementation class
           Adder server = new AdderImplements();

           // rmi registry within the server JVM with port number 1099
           LocateRegistry.createRegistry(1099);

           // Binds the remote object by the name service
           Naming.rebind("service", server);
           System.out.println("Server Started ");
       } catch (Exception e) {
           System.out.println(e.getMessage());
       }
   }
}
```

Client.java

```
import java.rmi.*;

public class Client {

   public static void main(String args[]) {
       try {
           String ip = "rmi://localhost/service";
           // lookup method to find reference of remote object
           Adder adder = (Adder) Naming.lookup(ip);
           System.out.println("sum: " + adder.add(1, 3));
       } catch (Exception e) {
           System.out.println(e.getMessage());
           e.printStackTrace();
       }
   }
}
```

Common Object Request Broker Architecture (CORBA)

CORBA, or **C**ommon **O**bject **R**equest **B**roker **A**rchitecture, is a standard architecture for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate.

CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware.

The CORBA is a standard defined by the **Object Management Group (OMG)** that enables software components written in multiple computer languages and running on multiple computers to work together.

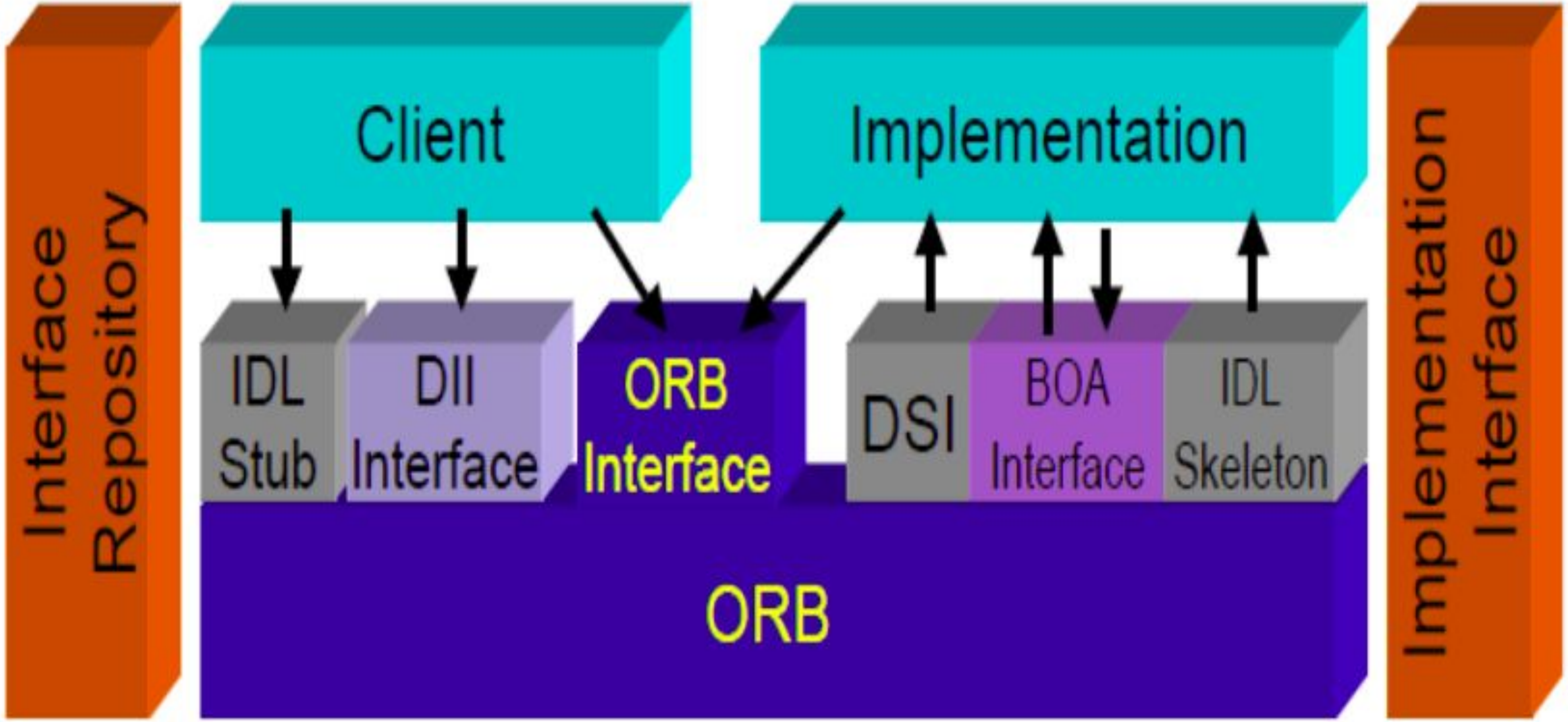
CORBA contd..

When introduced in 1991, CORBA defined the **Interface Design Language (IDL)** and **Application Programming Interface (API)**.

These allow client/server interaction within a specific implementation of an **Object Request Broker (ORB).**

The client send an ORB request to the SERVER/OBJECT implementation and this in turn, returns back either ORB Result or Error to the client.

CORBA Architecture



CORBA Architecture

ORB Interface: contains functionality that might be required by the clients or servers

Dynamic Invocation Interface (DII): used for dynamically invoking CORBA objects that were not known at implementation time.

Dynamic Skeleton Interface (DSI): helps to implement generic CORBA servants.

Basic Object Adapter (BOA): API used by the servers to register their object implementations.

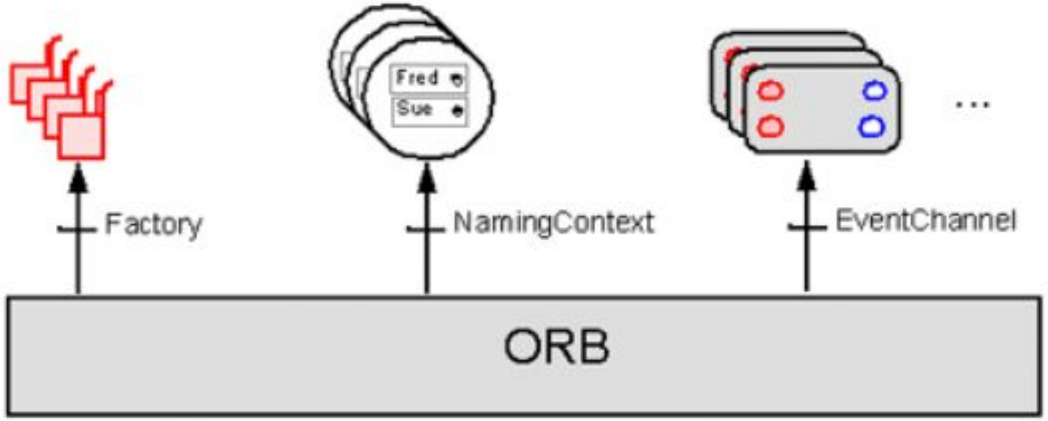
Interface Repository: provides type information necessary to issue requests using the DII.

Interfaces are defined in **OMG**'s **Interface Definition Language (IDL)**.

CORBA Services

Another important part of the CORBA standard is the definition of a set of distributed services to support the integration and interoperation of distributed objects.

As depicted in the graphic below, the services, known as **CORBA Services** or **COS**, are defined on top of the ORB. That is, they are defined as standard CORBA objects with IDL interfaces, sometimes referred to as "Object Services."


Fred
Sue
Factory
NamingContext
EventChannel
...
ORB