

Assignment 3

Roll no: 43144

Name: Ruchika Pande

Title: Distributed application using CORBA program.

Problem Statement:

To develop a calculator application with the CORBA program using JAVA IDL.

Objectives:

- To study about CORBA using JAVA

Theory:

CORBA

The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects. CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages, and operating systems. CORBA is essentially a design specification for an Object Request Broker (ORB), where an ORB provides the mechanism required for distributed objects to communicate with one another, whether locally or on remote devices, written in different languages, or at different locations on a network.

The CORBA Interface Definition Language, or IDL, allows the development of language and location-independent interfaces to distributed objects. Using CORBA, application components can communicate with one another no matter where they are located, or who has designed them. CORBA provides the location transparency to be able to execute these applications.

CORBA is often described as a "software bus" because it is a software-based communications interface through which objects are located and accessed.

Why CORBA

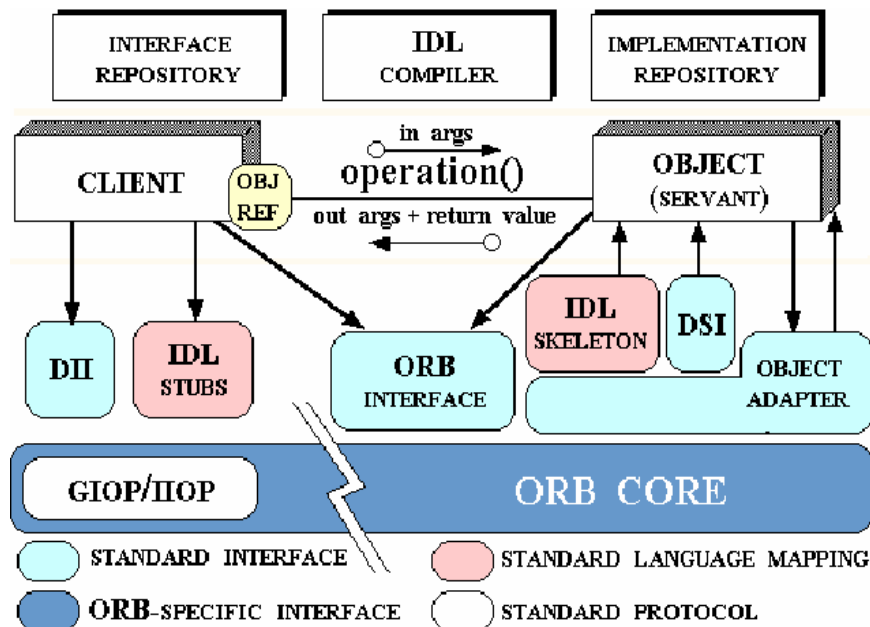
Distributed applications cause a lot of problems :

- Participating systems may be heterogeneous
- Access to remote services has to be location transparent
- Remote objects have to be found and activated
- State of objects has to be kept persistent and consistent

CORBA solves these problems as it:

- supports a remote method invocation paradigm
- provides location transparency
- allows to add, exchange, or remove services dynamically
- hides system details from the developer

Architecture of CORBA :



- **Object** -- This is a CORBA programming entity that consists of an *identity*, an *interface*, and an *implementation*, which is known as a *Servant*.
- **Servant** -- This is an implementation programming language entity that defines the operations that support a CORBA IDL interface. Servants can be written in a variety of languages, including C, C++, Java, Smalltalk, and Ada.
- **Client** -- This is the program entity that invokes an operation on an object implementation. Accessing the services of a remote object should be transparent to the caller. Ideally, it should be as simple as calling a method on an object, i.e., `obj->op(args)`. The remaining components in Figure 2 help to support this level of transparency.

- **Object Request Broker (ORB)** -- The ORB provides a mechanism for transparently communicating client requests to target object implementations. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.
- **ORB Interface** -- An ORB is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface described below.
- **CORBA IDL stubs and skeletons** -- CORBA IDL stubs and skeletons serve as the "glue" between the client and server applications, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler. The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimizations.
- **Dynamic Invocation Interface (DII)** -- This interface allows a client to directly access the underlying request mechanisms provided by an ORB. Applications use the DII to dynamically issue requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs (which only allow RPC-style requests), the DII also allows clients to make non-blocking *deferred* *synchronous* (separate send and receive operations) and *one way* (send-only) calls.
- **Dynamic Skeleton Interface (DSI)** -- This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.
- **Object Adapter** -- This assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects).

Java IDL

Java IDL is an Object Request Broker and a compiler, idlj, that maps IDL to the Java programming language. Java IDL can be used to define, implement, and access CORBA objects from the Java programming language. Java IDL is compliant with the CORBA/IIOP 2.2 Specification and the Java Language Mapping to OMG IDL (formal/99-07-59).

The Java IDL ORB supports transient CORBA objects - objects whose lifetimes are limited by their server process's lifetime. Java IDL also provides a transient name server to organize objects into a tree-directory structure. The name server is compliant with the Naming Service Specification. Transient objects and the name server are discussed later in this page.

No interface repository is provided as part of Java IDL. An interface repository is not required because under normal circumstances, clients have access to generated stub files.

Advantages of CORBA :

- **Maturity** - CORBA is extremely feature-rich, supporting many programming languages, operating systems, and a diverse range of capabilities such as transactions, security, Naming and Trading services.
- **Open standard** - users can choose the implementation from a variety of CORBA vendors.
- **Wide platform support** - CORBA implementations are accessible for a wide variety of computers, including IBM OS/390 and Fujitsu GlobalServer mainframes, LINUX, Windows and several embedded operating systems.
- **Wide language support** - CORBA defines standardized language mappings for a wide variety of programming languages, such as C, C++, Java, Smalltalk, COBOL, Python and IDLScript.
- **Efficiency** - The on-the-wire protocol infrastructure of CORBA guarantees that messages between clients and servers are transmitted in a compact representation.
- **Scalability** - The flexible, server-side infrastructure of CORBA makes it possible to develop servers that can scale from handling a small number of objects up to handling a virtually unlimited number of objects.

Disadvantages of CORBA :

- **Firewall unfriendly** - There's no real CORBA standard to bind an ORB and its clients to a port or a port range, there are (only) vendor specific options.
- **Regarded as complicated** – some remote invocation of CORBA interfaces is at least as simple as over XMLRPC (which is regarded as easy).
- No standard to get the initial reference for the naming services.
- **No official Perl making** - There are at least two Perl ORBs available as open source, but neither the mapping is official, nor are the implementations complete.

Implementation:

Defining IDL file:

- Define an IDL file which contains the required interfaces, methods, parameters, events, arguments, exceptions etc. within a module. The IDL file is saved as "Calc.idl".
- Map the IDL file into its equivalent JAVA file using the following commands:

```
idlj -fall Calc.idl
```

The above command leads to the formation of the following files within the package:

a)Calc.java

This file contains the JAVA version of the IDL interface.

b)_CalcStub.java

This file is the Client Stub.

c)CalcPOA.java

This file is the Server Skeleton.

d)CalcHelper.java

This Class provides the narrow() to cast the CORBA object references to the proper data types.

e)CalcHolder.java

This class is used to handle and provide functionalities to in/out/in-out statements.

f)CalcOperations.java

This Class contains all the methods defined in the IDL interface in its JAVA version.

Implement the Server:

The server side has two classes: Servant class and the Server class. Servant Class contains the implementations of the IDL interface. Server Class contains the main().

Server Class performs the following functionalities:

- A CORBA server needs a local ORB object. The server instantiates the ORB and registers its servant objects so that the ORB can find the server on client invocation.
- The ORB then obtains an initial reference to the Name Service. Reference to the root POA (Portable Object Adapter) is achieved and the POA manager is activated so that it becomes portable and can process requests.
- The server instantiates the servant objects to perform IDL interface defined operations.
- Common Object Services (COS) Naming Service is used to make the servant object's operations available to the clients.
- Obtain the Initial Naming Context. Pass the string "NameService" so that the ORB returns a naming context object.
- Use the narrow() to cast the object reference to its proper type.
- Register the servant with the Name Server by passing the string "Calc".
- Use the run() to wait for the client requests.

Implement the Client:

- Client class contains the main() and performs the following functionalities:
- A CORBA client needs a local ORB object. Hence the client needs to create and initialize the ORB.
- The client then uses the COS Naming Service to locate the Server.
- The client needs to get an Initial Naming Context by calling the resolve_initial_references().
- Narrow the object reference to its proper type by using the narrow().
- Invoke the required methods on the servant class as inputted by the client.

Running the application:

- Compile the server using the following command:

```
javac StartServer.java
```

- Compile the client using the following command:

```
javac StartClient.java
```

- Start the Name Service using the following command:

```
start orbd -ORBInitialPort 1050 -ORBInitialHost localhost
```

- Start the server using the following command:

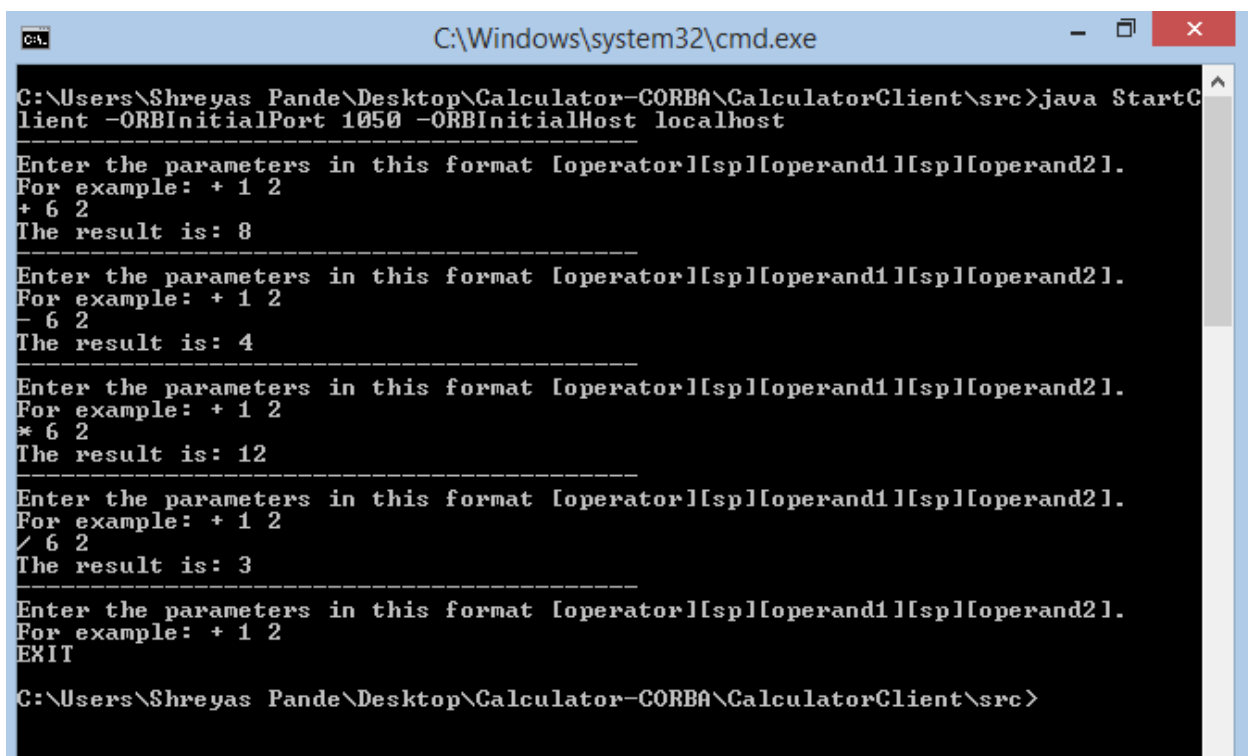
```
java StartServer -ORBInitialPort 1050 -ORBInitialHost localhost
```

- Start the Client using the following command:

```
java StartClient -ORBInitialPort 1050 -ORBInitialHost localhost
```

Output:

Client side:

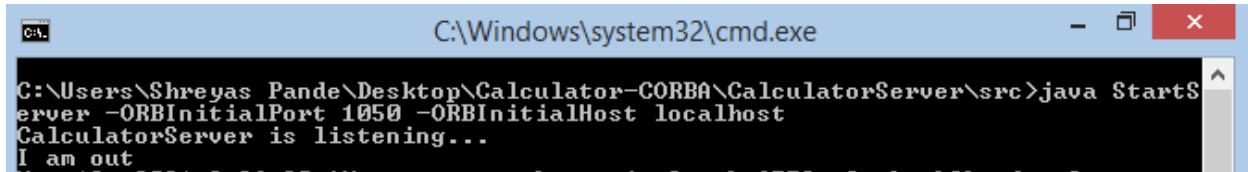


```
C:\Windows\system32\cmd.exe

C:\Users\Shreyas Pande\Desktop\Calculator-CORBA\CalculatorClient\src>java StartClient -ORBInitialPort 1050 -ORBInitialHost localhost
-----
Enter the parameters in this format [operator][sp][operand1][sp][operand2].
For example: + 1 2
+ 6 2
The result is: 8
-----
Enter the parameters in this format [operator][sp][operand1][sp][operand2].
For example: + 1 2
- 6 2
The result is: 4
-----
Enter the parameters in this format [operator][sp][operand1][sp][operand2].
For example: + 1 2
* 6 2
The result is: 12
-----
Enter the parameters in this format [operator][sp][operand1][sp][operand2].
For example: + 1 2
/ 6 2
The result is: 3
-----
Enter the parameters in this format [operator][sp][operand1][sp][operand2].
For example: + 1 2
EXIT

C:\Users\Shreyas Pande\Desktop\Calculator-CORBA\CalculatorClient\src>
```

Server side:



```
C:\Windows\system32\cmd.exe

C:\Users\Shreyas Pande\Desktop\Calculator-CORBA\CalculatorServer\src>java StartServer -ORBInitialPort 1050 -ORBInitialHost localhost
CalculatorServer is listening...
I am out
```

Conclusion:

In this assignment, we learned about creating distributed objects and establishing communication between them using CORBA. We used JAVA 8 with idlj compiler for the same. Thus, we successfully implemented this assignment in JAVA.