Group no. 13

Kajol Achhra-01, Priyanka Ahuja-04, Yogesh Tekwani - 68

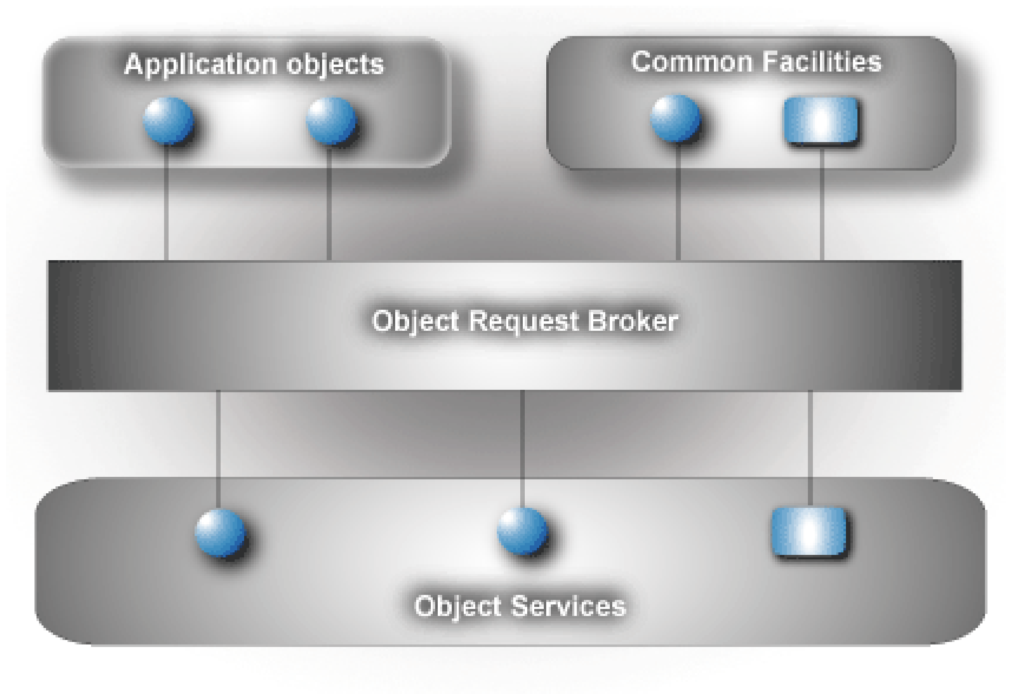
CORBA based ORB Interaction (Client - Server having different programming language implementation

1.INTRODUCTION:

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.

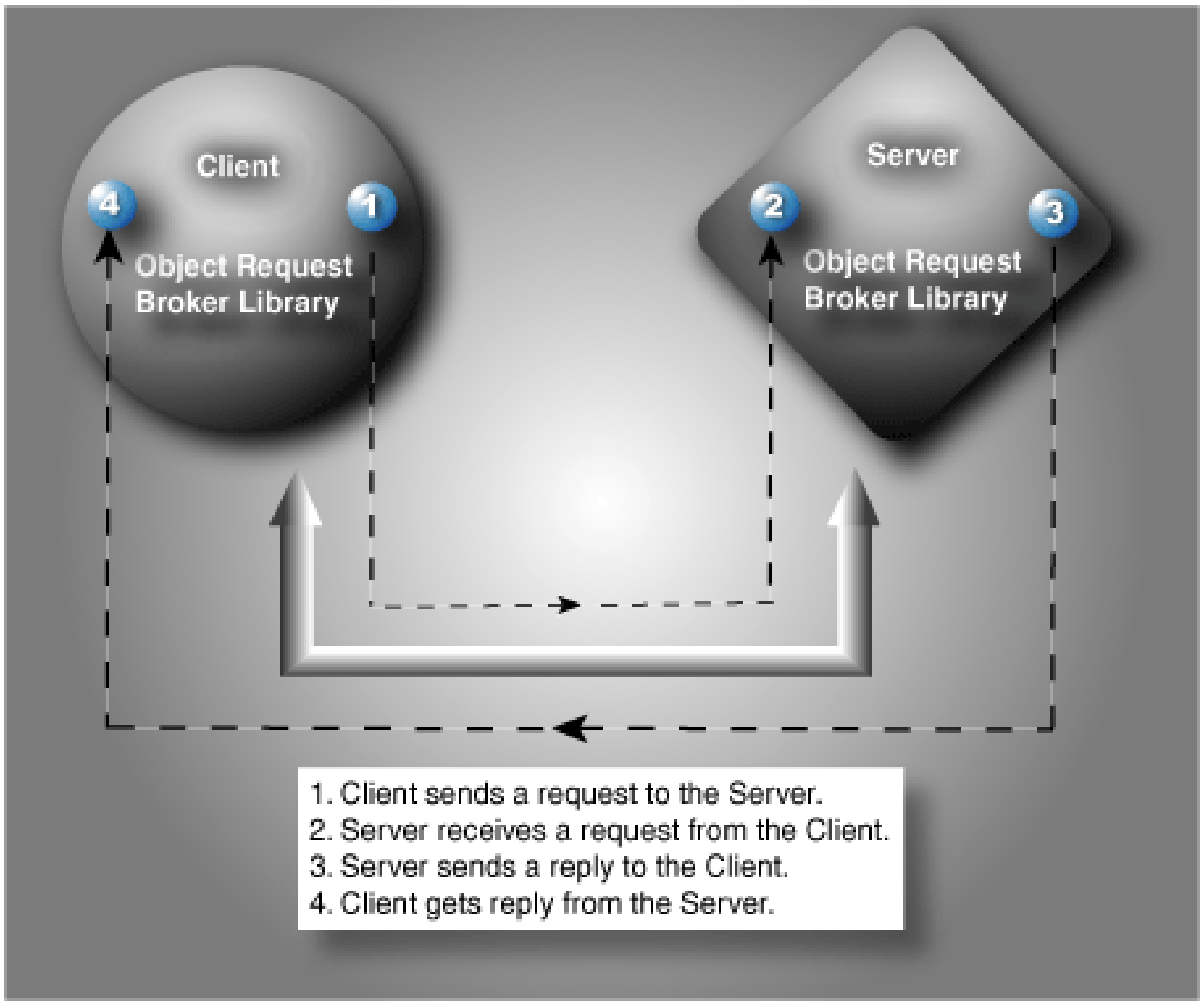
CORBA is often described as a "software bus" because it is a software-based communications interface through which objects are located and accessed. The illustration below identifies the primary components seen within a CORBA implementation.



Interface Definition Language (IDL)

A cornerstone of the CORBA standards is the Interface Definition Language. IDL is the OMG standard for defining language-neutral APIs and provides the platform-independent delineation of the interfaces of distributed objects. The ability of the CORBA environments to provide consistency between clients and servers in heterogeneous environments begins with a standardized definition of the data and operations constituting the client/server interface. This standardization mechanism is the IDL, and is used by CORBA to describe the interfaces of objects.

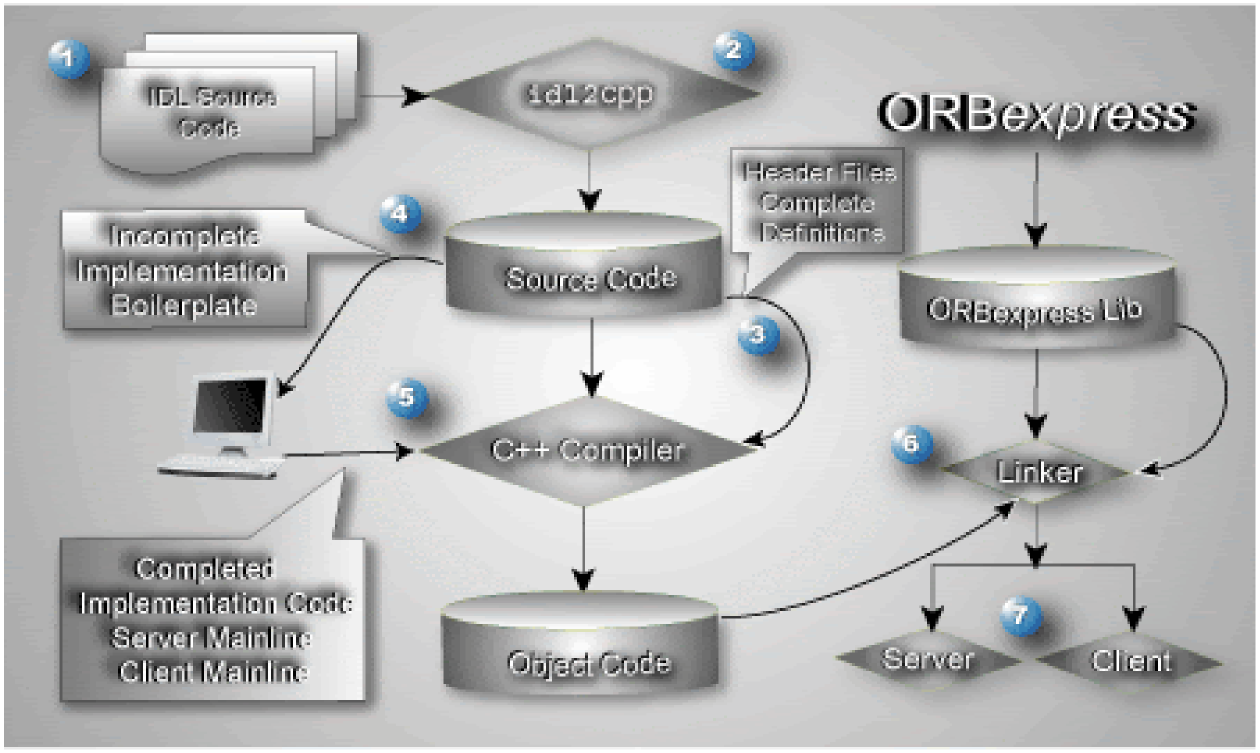
IDL defines the modules, interfaces and operations for the applications and is not considered a programming language. The various programming languages, such as Ada, C++, C#, or Java, supply the implementation of the interface via standardized IDL mappings.


Client
4
1
Object Request Broker Library
Server
2
3
Object Request Broker Library
1. Client sends a request to the Server.
2. Server receives a request from the Client.
3. Server sends a reply to the Client.
4. Client gets reply from the Server.

2. PRESENT YOUR DESIGN:

Application Development Using ORB*express*

The basic steps for CORBA development can be seen in the illustration below. This illustration provides an overview of how the IDL is translated to the corresponding language (in this example, C++), mapped to the source code, compiled, and then linked with the ORB library, resulting in the client and server implementation.


1
IDL Source Code
2
idl2cpp
ORBexpress
Header Files Complete Definitions
4
Incomplete Implementation Boilerplate
Source Code
ORBexpress Lib
3
5
C++ Compiler
6
Linker
Completed Implementation Code Server Mainline Client Mainline
Object Code
7
Server
Client

3. Explain the working concept of your applications and layered architecture of communication protocol

Steps to implement java CORBA ORB based interaction

Create the IDL to Define the Application Interfaces

The IDL provides the operating system and programming language independent interfaces to all services and components that are linked to the ORB. The IDL specifies a description of any services a server component exposes to the client. The term "IDL Compiler" is often used, but the IDL is actually translated into a programming language.

Translate the IDL

An IDL translator typically generates two cooperative parts for the client and server implementation, stub code and skeleton code. The stub code generated for the interface classes is associated with a client application and provides the user with a well-defined Application Programming Interface (API). In this example, the IDL is translated into C++.

Compile the Interface Files

Once the IDL is translated into the appropriate language, C++ in this example, these interface files are compiled and prepared for the object implementation.

Complete the Implementation

If the implementation classes are incomplete, the spec and header files and complete bodies and definitions need to be modified before passing through to be compiled. The output is a complete client/server implementation.

Compile the Implementation

Once the implementation class is complete, the client interfaces are ready to be used in the client application and can be immediately incorporated into the client process. This client process is responsible for obtaining an object reference to a specific object, allowing the client to make requests to that object in the form of a method call on its generated API.

Link the Application

Once all the object code from steps three and five have been compiled, the object implementation classes need to be linked to the C++ linker. Once linked to the ORB library, in this example, ORB*express*, two executable operations are created, one for the client and one for the server.

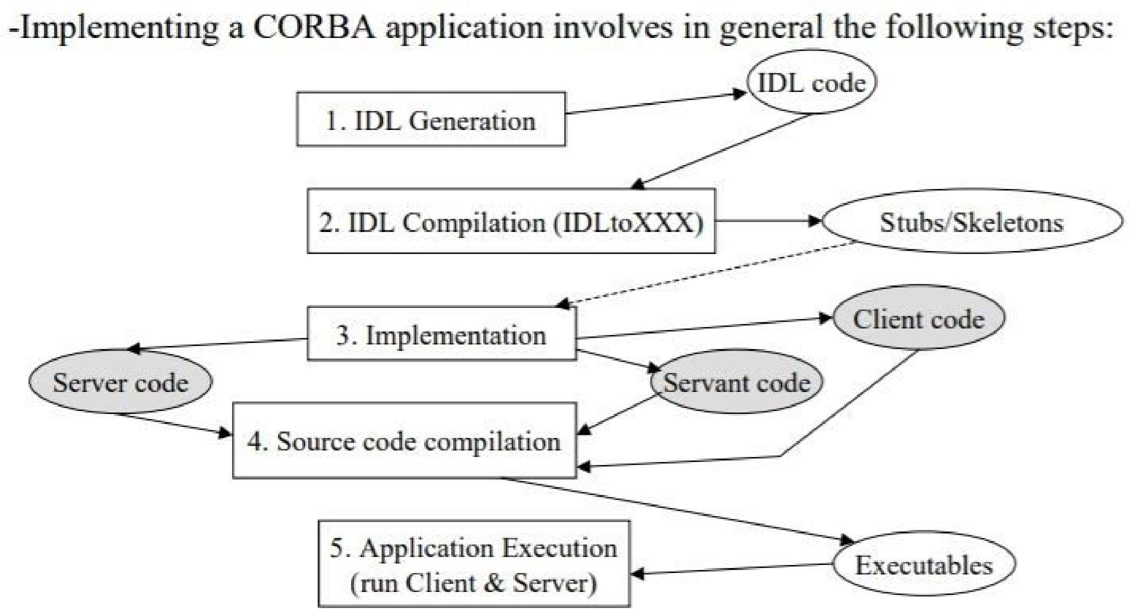
Run the Client and Server

The development process is now complete and the client will now communicate with the server. The server uses the object implementation classes allowing it to communicate with the objects created by the client requests.

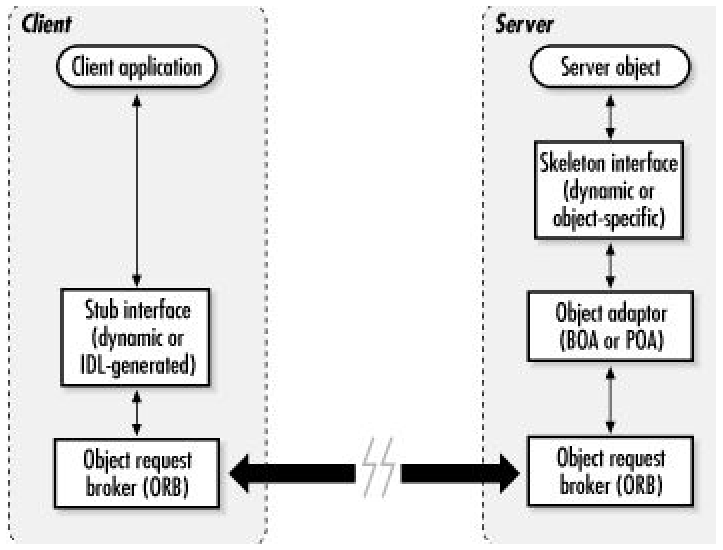
In its simplest form, the server must perform the following:

- Create the required objects.
- Notify the CORBA environment that it is ready to receive client requests.
- Process client requests by dispatching the appropriate servant.

4. Explain and justify the CORBA based ORB interaction (client - server in different languages)

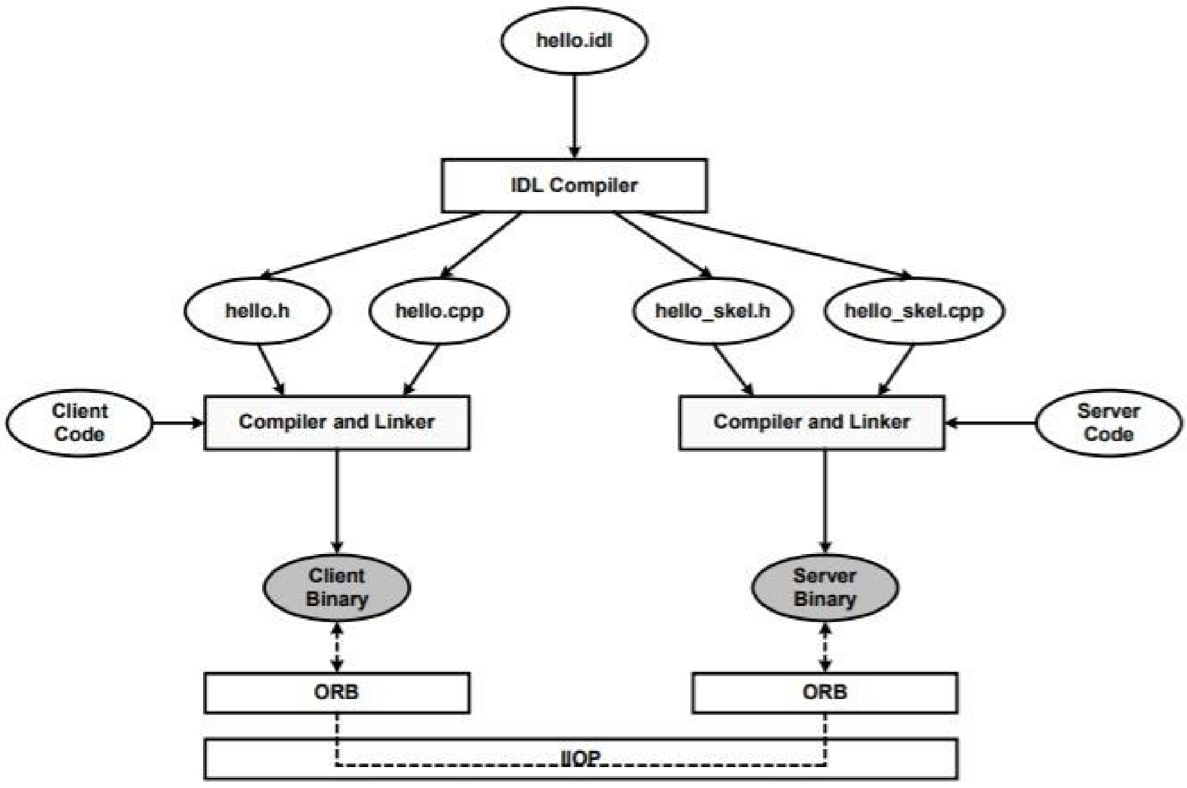

-Implementing a CORBA application involves in general the following steps:
1. IDL Generation
IDL code
2. IDL Compilation (IDLtoXXX)
Stubs/Skeletons
3. Implementation
Client code
Server code
Servant code
4. Source code compilation
5. Application Execution (run Client & Server)
Executables

The servant corresponds to the remote CORBA Object Class
The Client and server may either be implemented in the same language or in the different languages like C Java. CORBA applications can interact, regardless of language they are written in.


Client
Client application
Stub interface (dynamic or IDL-generated)
Object request broker (ORB)
Server
Server object
Skeleton interface (dynamic or object-specific)
Object adaptor (BOA or POA)
Object request broker (ORB)

A. **Client - server design (stateful/stateless)**
B. **Server Creation Semantic**
C. **Persistent and transient communication**
D. **Synchronous / Asynchronous communication**
E. **Call semantics**
F. **Concurrent access to multiple servers**
G. **Reply caching of idempotent**

IDL to C++ Compiler Functionality



Communication model:

CORBA Approach to Client/Server Development

The CORBA model provides a more flexible approach to developing distributed applications. The CORBA model:

- Formally separates the client and server portions of the application

  A CORBA client application knows only how to ask for something to be done, and a CORBA server application knows only how to accomplish a task that a client application has requested it to do.

Because of this separation, developers can change the way a server accomplishes a task without affecting how the client application asks for the server application to accomplish the task.

- Logically separates an application into objects that can perform certain tasks, called operations

  CORBA is based on the distributed object computing model, which combines the concepts of distributed computing (client and server) and object-oriented computing (based on objects and operations).

  In object-oriented computing, objects are the entities that make up the application, and operations are the tasks that a server can perform on those objects. For example, a banking application could have objects for customer accounts, and operations for depositing, withdrawing, and viewing the balance in the accounts.

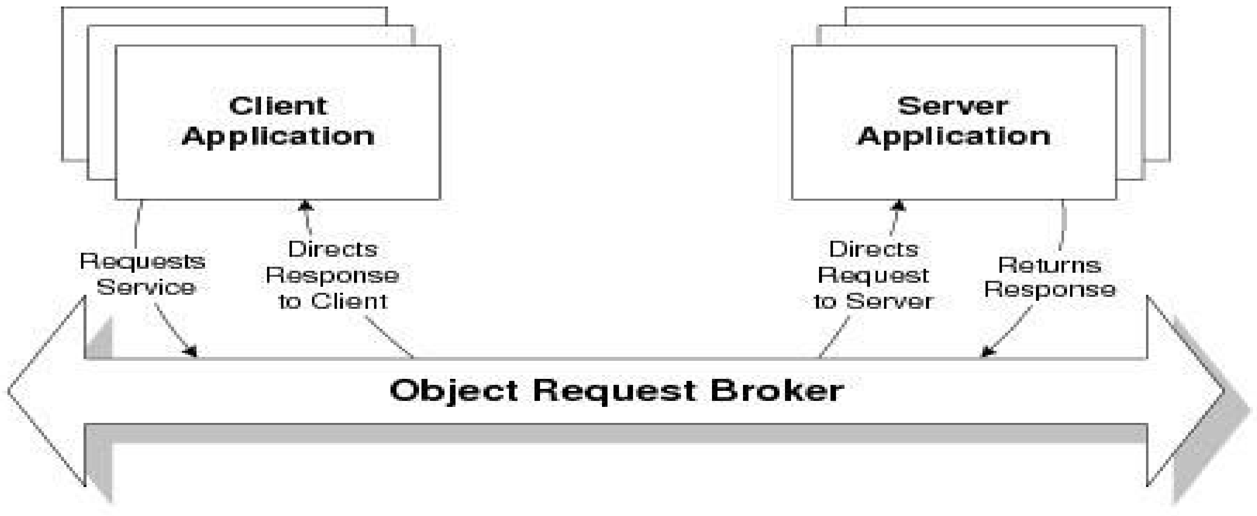
- Provides data marshaling to send and receive data with remote or local machine applications

  For example, the CORBA model automatically formats for big or little endian as needed. (Refer to the preceding section for a description of data marshaling.)

- Hides network protocol interfaces from the applications

  The CORBA model handles all network interfaces. The applications see only objects. The applications can run on different machines and, because all the network interface code is handled by the ORB, the application does not require any network-related changes if it is later deployed on a machine that supports a different network protocol.

The CORBA model allows client applications to make requests to server applications, and to receive responses from them without direct knowledge of the information source or its location. In a CORBA environment, applications do not need to include network and operating system information to communicate; instead, client and server applications communicate with the Object Request Broker (ORB). The following figure shows the ORB in a client/server environment.



CORBA defines the ORB as an intermediary between client and server applications. The ORB delivers client requests to the appropriate server applications and returns the server responses to the requesting client application. Using an ORB, a client application can request a service without knowing the location of the server application or how the server application will fulfill the request.

In the CORBA model, client applications need to know only what requests they can make and how to make the requests; they do not need to be coded with any implementation details of the server or of the data formats. Server applications need only know how to fulfill the requests, not how to return data to the client application.

This means that programmers can change the way a server application accomplishes a task without affecting how the client application asks for the server application to accomplish that task. For example, as long as the interfaces between the client and the server applications do not change, programmers can evolve and create new implementations of a server application without changing the client application; in addition, they can create new client applications without changing the server applications.