

# Introduction to CORBA

- CORBA (Common Object Request Broker Architecture) provides a framework for the development and execution of distributed applications.
- There may be three types of distribution in a system, *data*, *computation* or *users*.
- For some applications, data may be distributed due to administrative, ownership or historical reasons. The owner may permit the data to be accessed remotely, e.g., stock prices.
- Some applications execute on multiple machines in order to take advantage of parallel computation.
- Some applications execute on multiple computers because the users of the application are geographically distributed and they interact with each other through the application.

## Some issues related to distributed systems

- The communication between objects which are within the same process are much faster compared to communication between objects on different machines.
- So it is usually bad to design distributed applications when multiple distributed objects have very tight interactions.
- If two objects are co-located, they fail together, but distributed objects fail independently. Hence, a designer has to take into account how to deal with failure of distributed objects.
- When two objects are co-located in the same process, we need not be concerned with security. However, when objects are distributed over several machines, we need security mechanisms to authenticate objects.

## What is CORBA?

- *Distributed object systems* are distributed systems in which all entities are modelled as objects.
- The Common Object Request Broker Architecture (CORBA) is a standard architecture for distributed object systems. It allows a distributed and heterogeneous collection of objects to interoperate.
- The Object Management Group (OMG) comprises of over 700 companies and organizations and OMG is responsible for defining CORBA.
- The basic CORBA paradigm is that of a *request for services of a distributed object*. In other words, an object can request for the services of a second object irrespective of where this second object is.
- The services that an object provides are given by its *interface*. Interfaces are defined in OMG's Interface Definition Language (IDL).
- Distributed objects are identified by *object references* and the object references are assigned types by the IDL interfaces.

## Object Request Broker (ORB)

- The ORB is the distributed service that implements the request to the remote object.
- It locates the remote object on the network, communicates the request to the object, waits for the results and when available communicates those results back to the client.
- The ORB implements *location transparency*. Exactly the same request mechanism is used by the client and the CORBA object regardless of where the object is located.
- The ORB implements programming language independence for the request. The client issuing the request can be written in a different programming language from the implementation of the CORBA object.
- The ORB does the necessary translation between programming languages.

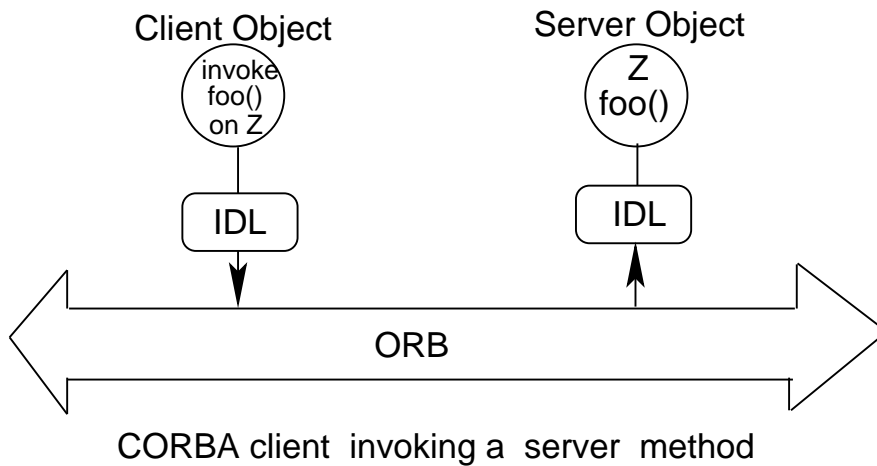
# CORBA as a Standard for Distributed Objects

- The CORBA specification defines an API for clients of a distributed object as well as an API for the implementation of a CORBA object.
- This means that code written for one vendor's CORBA product could be rewritten (with little effort) to work with a different vendor's product.
- In reality, CORBA clients are portable, but object implementations need some rework to port from one CORBA product to another.
- CORBA 2.0 defines a network protocol called *Internet Inter-ORB Protocol* (IIOP) that allows clients using a CORBA product from any vendor to communicate with objects using a CORBA product from any other vendor.
- IIOP works across the internet, or more precisely across any TCP/IP implementation.

## CORBA Architecture

- CORBA can be conceptualized as a communication bus for client-server objects. If object **A** invokes a method on object **B**, **A** is the client and **B** is the server.
- Server interfaces must be specified in the CORBA standard *Interface Definition Language* (IDL). An IDL interface description is then mapped using an IDL compiler to native language bindings such as Java, C++ etc.
- For example, a server object implemented in C++ can be accessed by a Java applet. The applet programmer invokes methods on the server as though they are local Java method calls.
- The ORB is the mediator, responsible for brokering interactions between objects. Its job is to provide object location and access transparency, by facilitating client invocations of methods on server objects.
- A client can connect or bind to a server object statically if the server interface is known at build time.
- Alternatively, the client can use dynamic binding to ascertain the server interface and construct a call to the server.

# Server Side



- The server side of CORBA begins with the specification of an interface in IDL. IDL is an object-oriented declarative language for specifying server interfaces.
- However, IDL is not a programming language, i.e., server implementations are not written in IDL. Rather, IDL interface descriptions are mapped to a programming language for implementing the server.

## Server Side

- A few IDL features,
  - an **interface** corresponds to a class.
  - a property is defined as an **attribute**, which is mapped by the IDL compiler to get and set methods. A **readonly** attribute maps to a get method.
  - An **operation** corresponds to a method.
  - parameters must be specified as **in**, **out** or **inout**.
- An example :

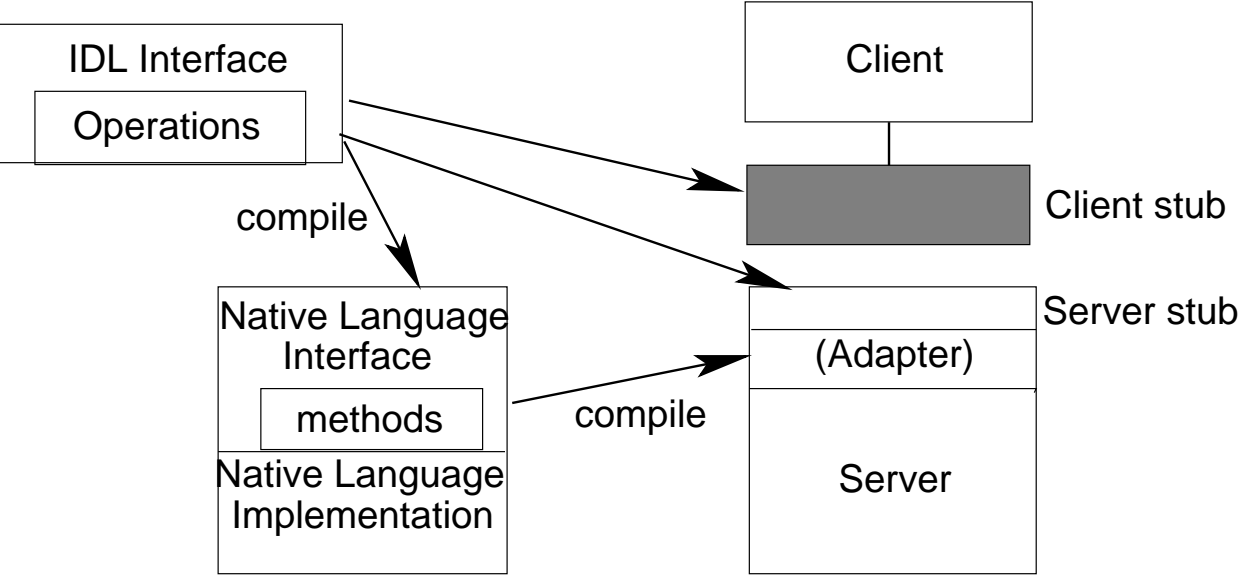
```
// IDL
interface Account
{
    //Attributes
    attribute float balance;
    readonly attribute string owner;

    //Operations
    void makeDeposit(in float amount,
                    out float newBalance);
    void makeWithdrawal(in float amount,
                       out float newBalance);
};
```

## Server Side

- An IDL compiler is used to generate a *server stub*, also referred to as a *skeleton*, which gets linked to the server program.
- The server stub provides static interfaces to call methods of an object implementation. It unmarshals methods and parameters that come from the client via the ORB.
- The IDL compiler also generates a native language interface for implementing the server, as well as a client side stub.
- An important component of the ORB is the *object adapter*, which is responsible for managing server objects and related object references.
- It uses the CORBA *implementation repository* to locate a server name, identify the activation mode and determine the launch and access permissions.
- Server *activation modes* define the run-time characteristics of a server, including whether a server process can contain one or more objects and if multiple clients are permitted to access the same object.

# Generation of stubs from IDL description



## Client Side

- The IDL compiler generates a *client stub* that gets linked to program which wishes to statically invoke a server method through the associated interface.
- The *client stub* maps a CORBA server side object to a native object in the client's language.
- It acts as a proxy for remote server objects and marshals methods and parameters to be transmitted via the ORB.
- CORBA also supplies the *dynamic invocation interface* (DII) for client programs to discover server interfaces at run-time.
- The CORBA *interface repository* (IFR) contains compiled IDL descriptions that can be interrogated via the DII.
- A client can locate a server object by obtaining an object reference directly from a server, by requesting an object by name either by,
  - via the CORBA naming service, or by
  - asking the CORBA trader service to return references to objects that match some criteria.

## CORBA Process

We can create a static CORBA server and client through the following steps.

- Specify the server interface in IDL.
- Run the IDL description through an IDL compiler. This generates a native language interface, server stub, and client stub. The server and client stubs can be for different programming languages.
- During the above step, the IDL compiler can optionally write a compiled interface description to the interface repository.
- Next, implement the server.
- Compile the server program and link in the server stub that was generated by the IDL compiler. The result is an executable server program that can accept method invocations via CORBA.

## CORBA Process (continued)

- Register the server in the implementation repository, specifying server name, activation mode, and launch and access permissions.
- Implement the client. The programmer writes server object method calls as though they are local methods. The programmer uses the syntax and conventions of the client's native programming language.
- Compile the client program and link in the client stub that was generated by the IDL compiler.
- When the client is executing, it uses the ORB to bind to the server object and obtain an object reference.
- Using the object reference, the client invokes server object methods.