Chapter 6. CORBA-based Architecture

6.1 Introduction to CORBA
6.2 CORBA-IDL
6.3 Designing CORBA Systems
6.4 Implementing CORBA Applications

Chapter 6. CORBA-based Architecture

Part 6.1 Introduction to CORBA

1. Introduction
2. Distributed Architecture
3. Middleware Systems
4. CORBA Architecture

1. Introduction

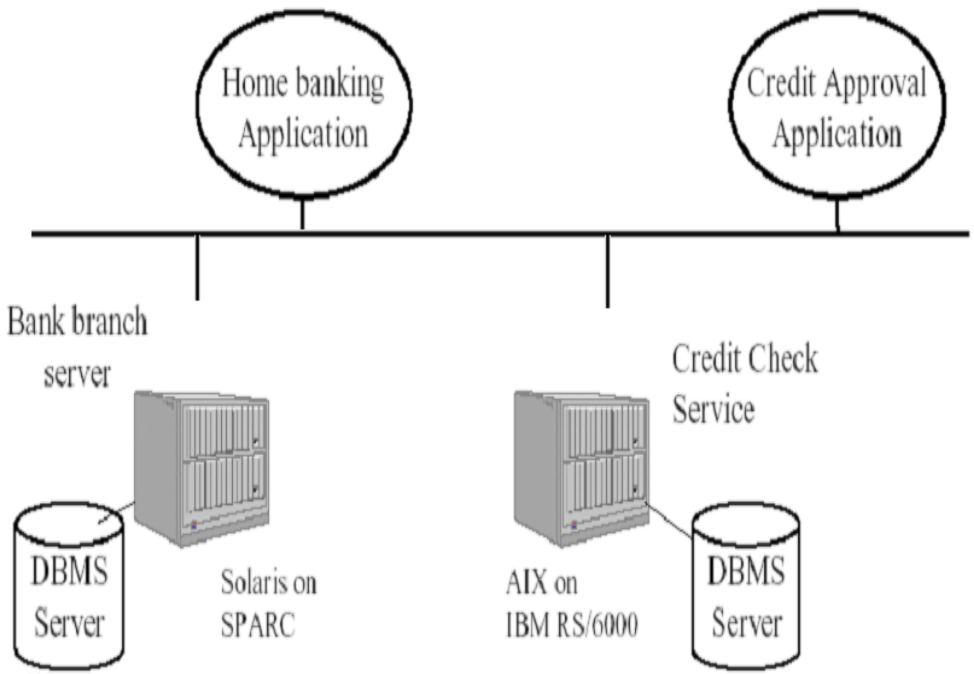
- CORBA is a software standard that is defined and maintained by the Object Management Group (OMG).
- The OMG:
  - -Founded in 1989 by eight companies as a non-profit organization.
  - -The consortium now includes over 800 members.
  - -Charter: establishment of industry guidelines and detailed object management specifications to provide a common framework for application development.
- OMG produces specifications, not implementations
  - -Implementations of OMG specifications can be found on over 50 operating systems
- CORBA is the acronym for *Common Object Request Broker Architecture*. It consists of a standard framework for developing and maintaining ***distributed software systems***. Specifically, it provides
  - -A **RPC mechanism** allowing the invocation of operations across different programming languages, hardware, and operating system platforms, achieving **portability** and **interoperability**.
  - -A **component model**, the CORBA Component Model (CCM), for **reusable** component development.

2. Distributed Architecture

Definition

A distributed architecture is an architecture supporting the development of applications and services that can exploit a physical architecture consisting of multiple, autonomous processing elements. Those elements do not share primary memory but cooperate by sending messages over the network.

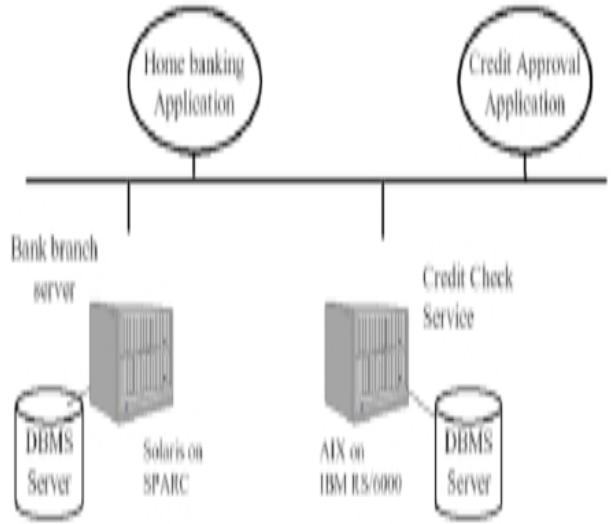
Example-Distributed System



Key Characteristics:

- Multiple autonomous components
- Components are not shared by all users
- Resources may not be accessible
- Software runs in concurrent processes on different processors
- Multiple points of control
- Multiple points of failure

Underlying Issues



-Some of the key challenges involved in designing, implementing and operating a distributed system include:

÷Heterogeneity

-Heterogeneous hardware, OS, languages, protocols etc.

÷Concurrency

-Resource sharing and concurrent access to data pose the issue of data integrity.

÷Failure

-Independent failure: often we want the system to keep working after one or more have failed.
-Unreliable communication: connections may be unavailable; messages may be lost.

÷Insecure communication

-The interconnections among the computers may be exposed to unauthorized eavesdropping and message modification.

÷Costly Communication

-The interconnections among the computers usually provide lower bandwidth, higher latency and higher cost communication compared to independent processes in a single machine

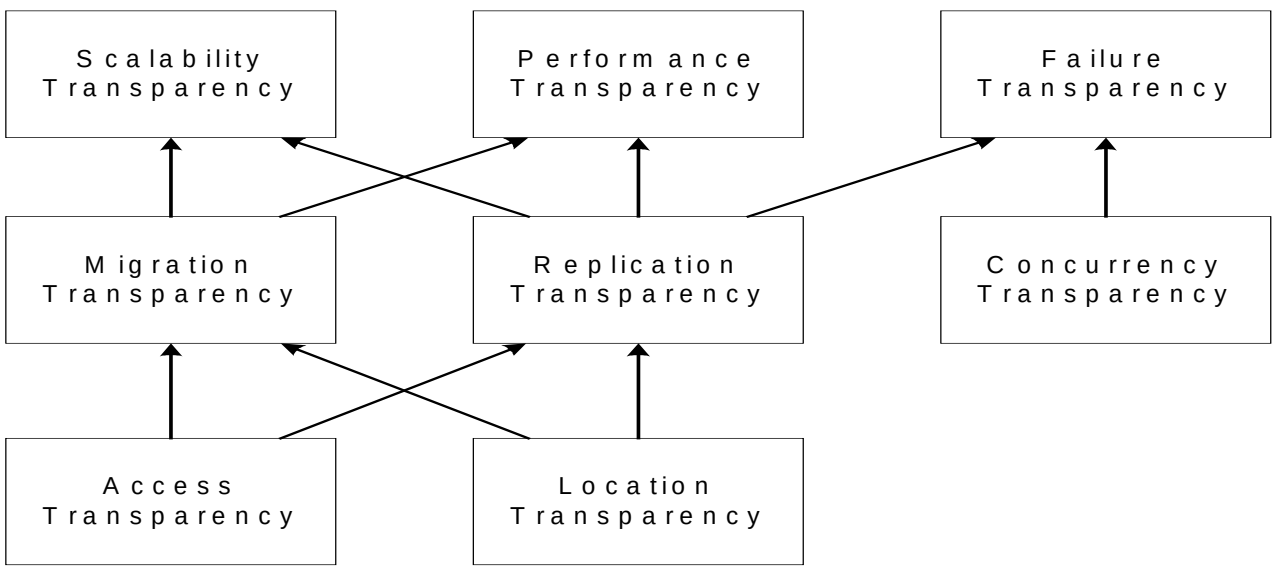
Transparency

- Distributed systems should be perceived by users and application programmers as a ***whole*** rather than as a ***heterogeneous*** collection of cooperating components: this is referred to as ***transparency***.

  -Transparency has different dimensions that were identified by ANSA as part of the International Standard of Open Distributed Processing (ODP).

  -These represent various properties that distributed systems should have.

Dimensions of Transparency in Distributed Systems

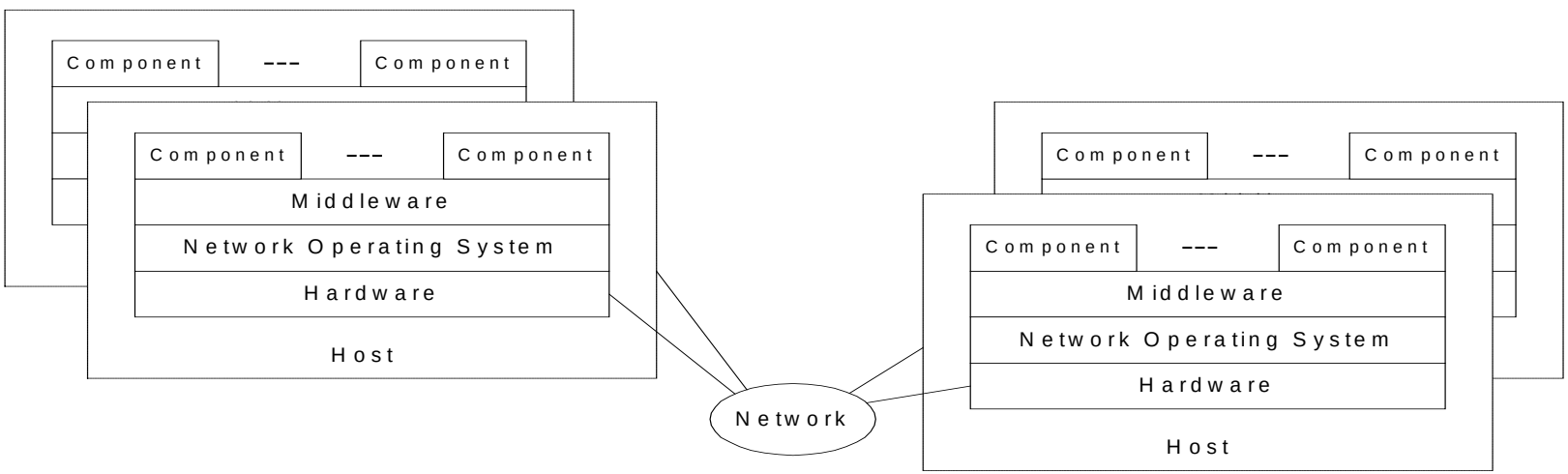


3. Middleware Systems

Definition: *Middleware is software that enables interprocess communication. It provides an API that isolates the application code from the underlying network communication formats and protocols (FAPs).*

-Middleware systems implement the various forms of distribution transparencies by creating the *illusion of unity and homogeneity* within the network, what is called in other words the ***"single-system image".***

÷They act as glue between autonomous components and processes (e.g., clients, server) by providing generic services on top of the OS.



-There are three kinds of middleware systems: transaction-oriented middleware, message-oriented middleware, and object-oriented middleware.

÷ *Transaction-oriented middleware* supports distributed computing involving database applications.

÷ *Message-oriented middleware* supports reliable, asynchronous communications among distributed components.

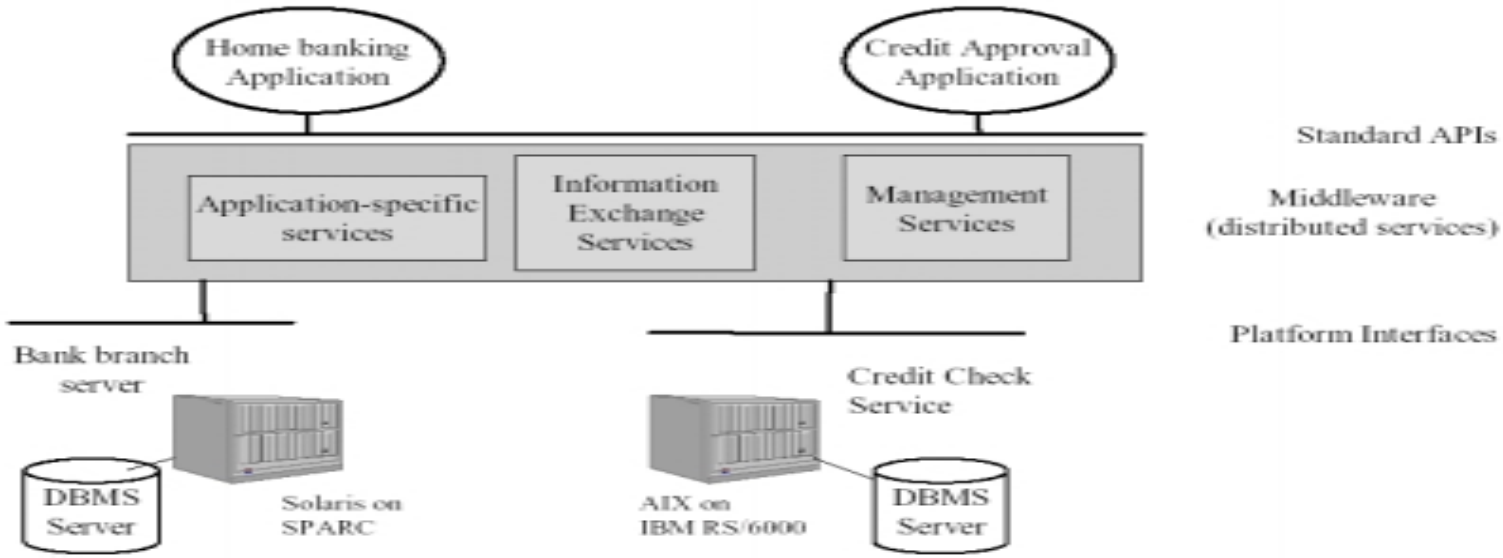
÷*Object-oriented middleware* systems are based on object-oriented paradigm, and primarily supports synchronous communications among distributed components.

-The most popular object-oriented middleware paradigms include *CORBA*, *DCOM*, *DotNET*, and *EJB* (which is based on RMI).

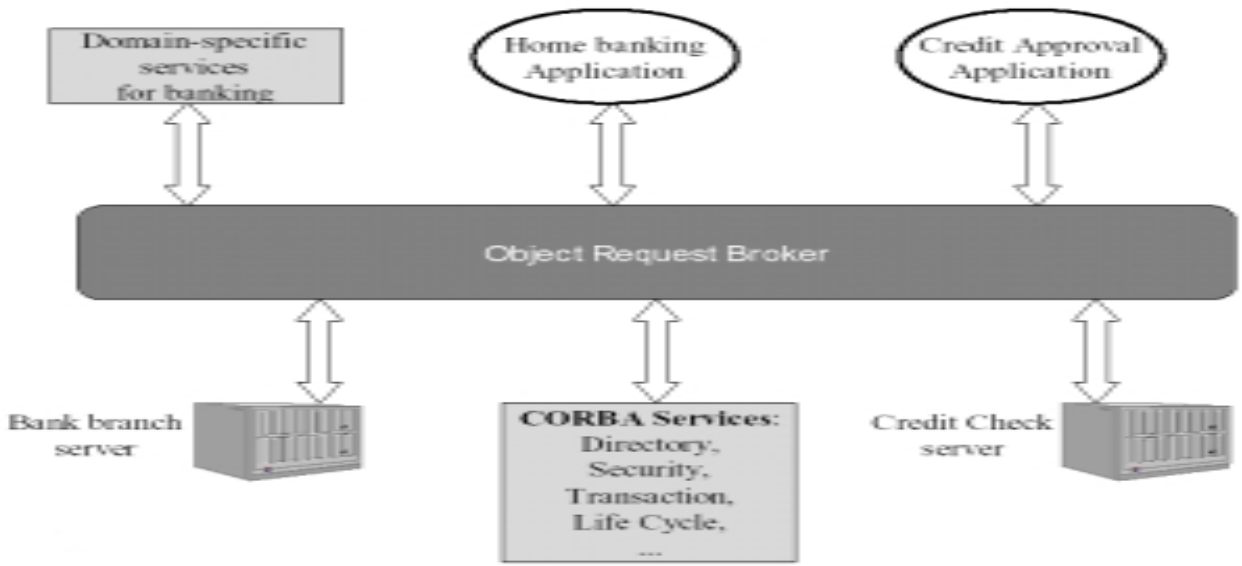
÷All these middleware systems, also referred to as Object-Oriented middleware, are based on the *Remote Procedure Call (RPC)* framework foundation.

÷ They extend RPC framework by introducing object-oriented mechanisms.

Example - Middleware



Example - Distributed Middleware with CORBA



Remote Procedure Call (RPC)

-RPC allows the invocation of operations across different hardware and operating system platforms.

÷ Provide the same mechanism as local procedure call but at the interprocess level.

÷ Support a common Interface Definition Language (IDL).

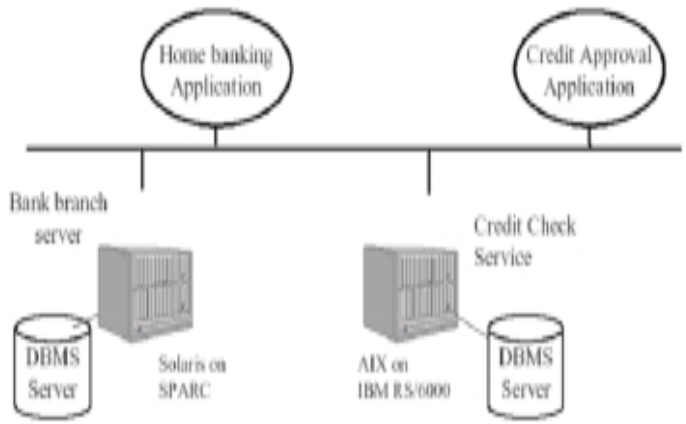
-The RPC mechanism is provided by a RPC software, which handles transparently all the steps involved. The functionality of the RPC software include:

÷ Location of the server functions, and and concurrent requests.

÷ Parameters passing and data representation.

÷ Failure management

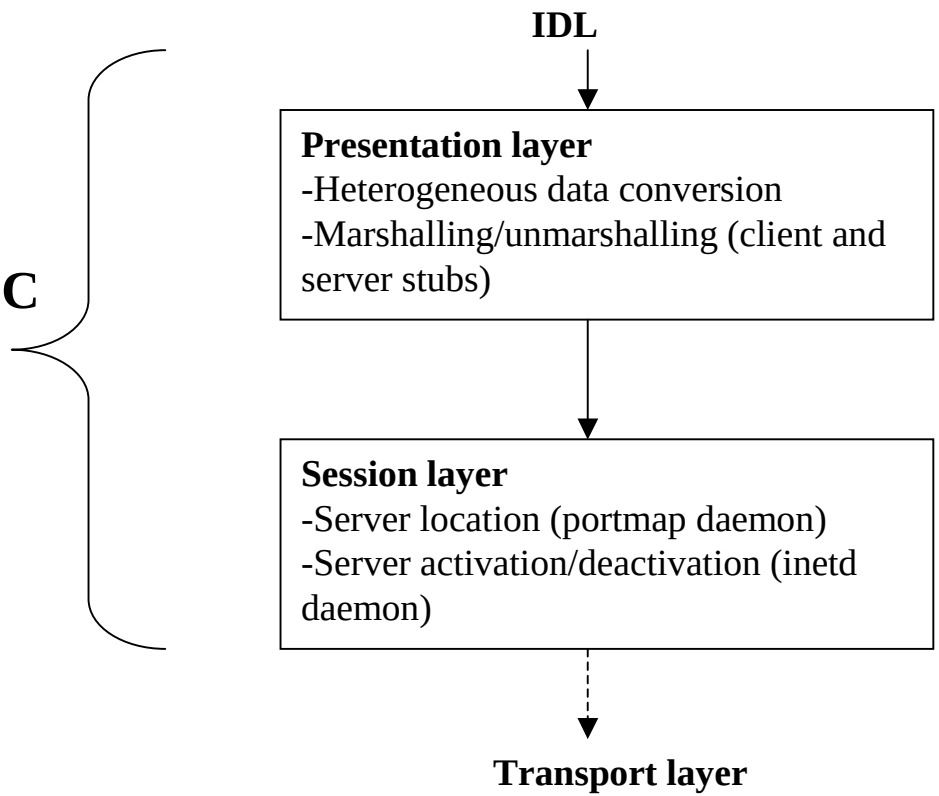
÷ Security management



-The RPC software provides an implementation of the session and presentation layer. Note that the transport layer (below) is implemented as a socket.

-The *presentation layer* implementation enables data conversion and formatting (e.g., marshalling/unmarshalling).

-The *session layer* implementation enables clients to locate RPC servers statically or dynamically, and activates or deactivates RPC servers when requests arrive.

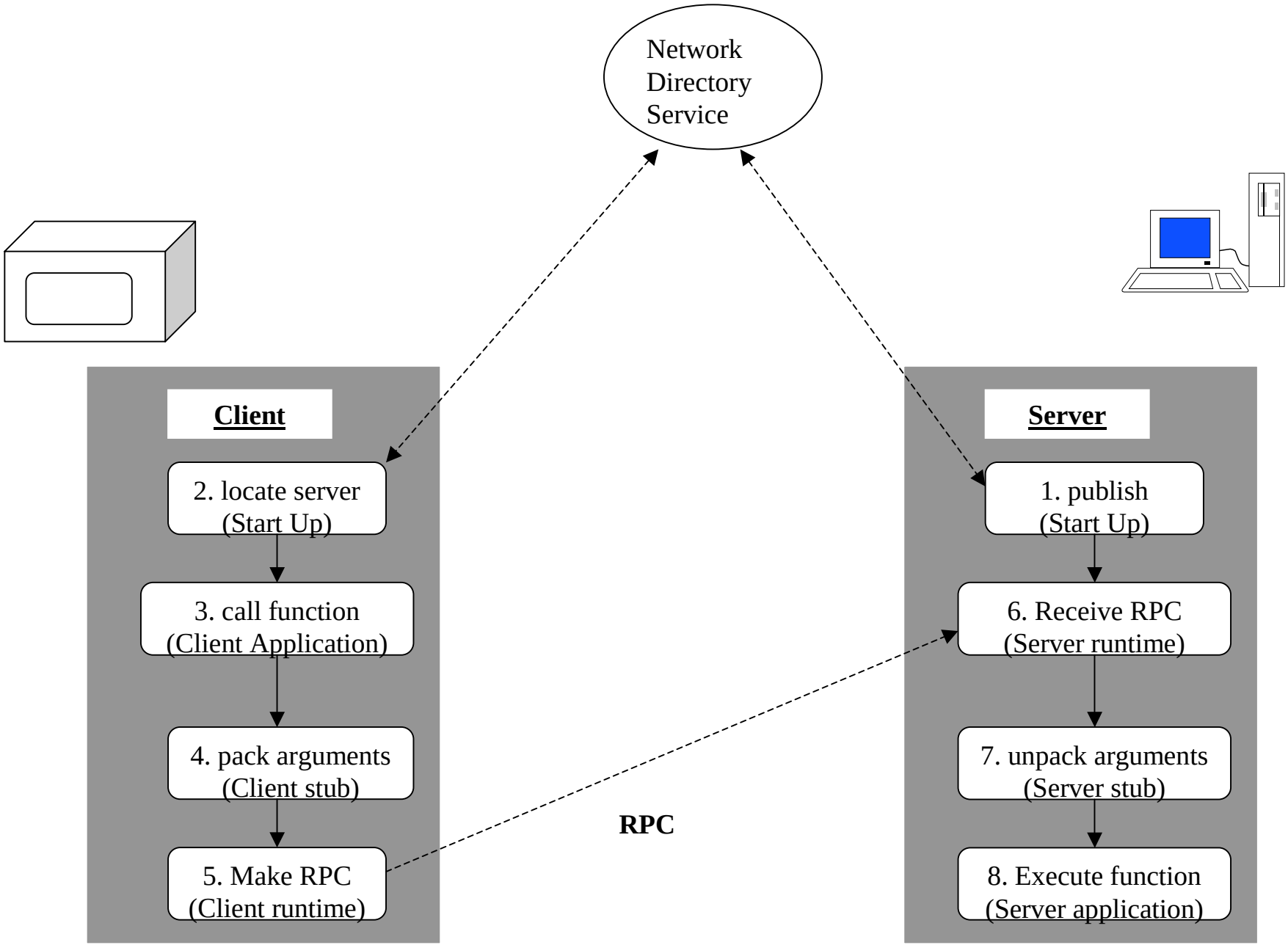


With OO middleware such as CORBA:

-The *presentation layer* fulfills the same functions as corresponding layer for RPC. It is also in charge of mapping object references to suitable format for the transport layer.

-The *session layer* is in charge of mapping object references to hosts, activating and deactivating objects, executing the requested services, and synchronizing client and server.

RPC Mechanism



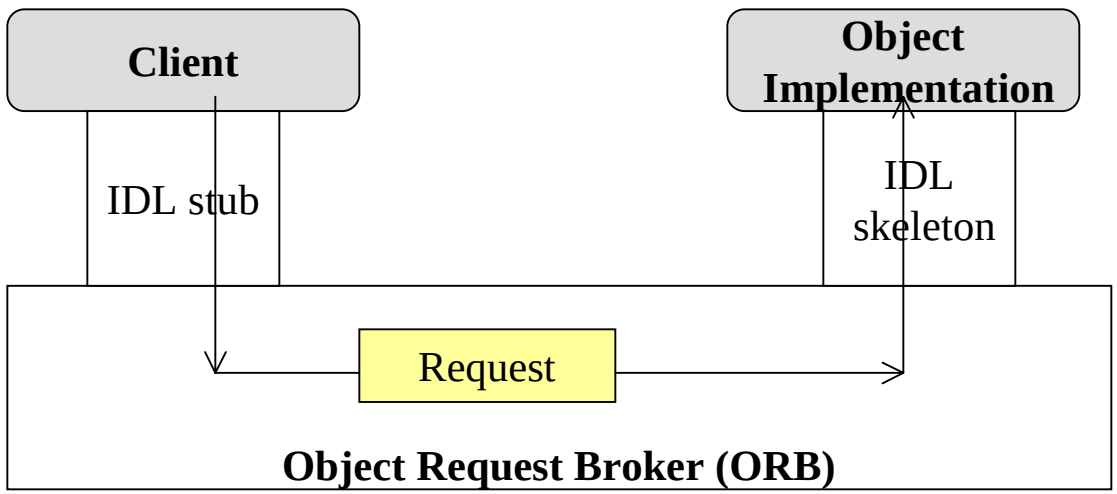
4. CORBA Architecture

CORBA in a Nutshell

÷In CORBA the services that an object provides are expressed in a contract that serves as the interface between it and the rest of the system.

÷The Object interface is expressed using a special language named *Interface Definition Language* (IDL).

÷For objects to communicate across the network, they need a communication infrastructure named *Object Request Broker* (ORB).

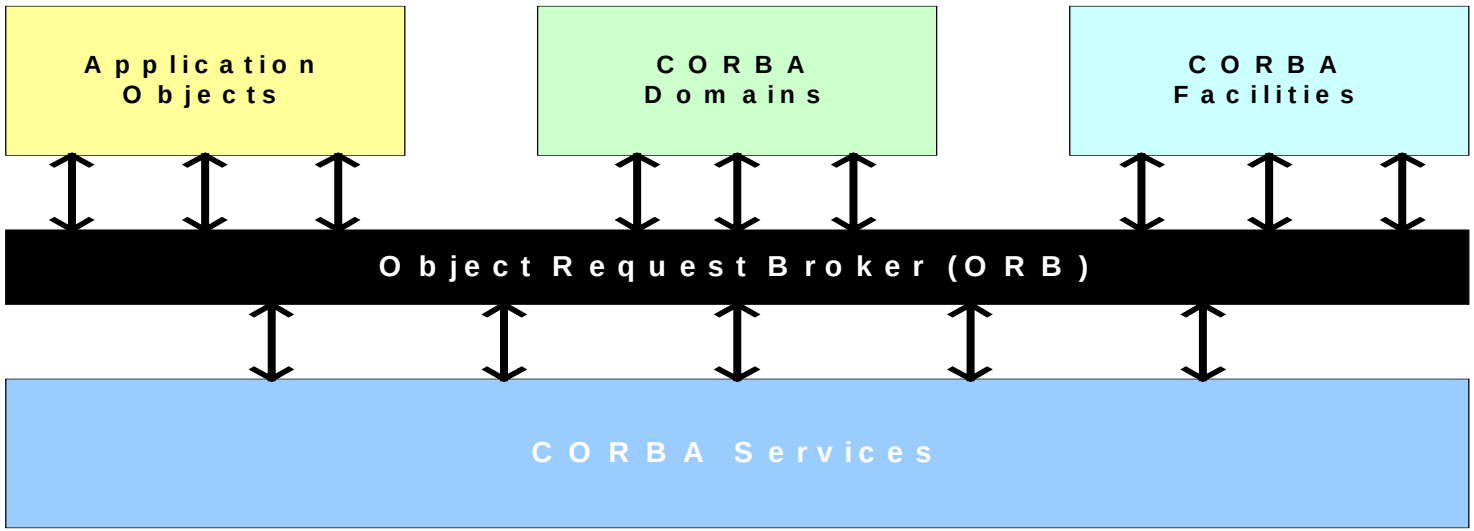


÷Both client and object implementation are isolated from the ORB by an IDL interface. Clients see only the object's interface, never the implementation.

÷To communicate, the request does not pass directly from client to object implementation; instead every request is passed to the client's local ORB, which manages it.

CORBA Reference Model Architecture

-The CORBA standard relies on a reference model named the *Object Management Architecture (OMA).*



- A typical CORBA implementation includes:

• An Object Request Broker (ORB) implementation
• An Interface Definition Language (IDL) compiler
• Implementations of Common Object Services (COS), also called CORBA Services
• Common Frameworks, also called CORBA facilities
• An Internet Inter ORB Protocol (IIOP) implementation

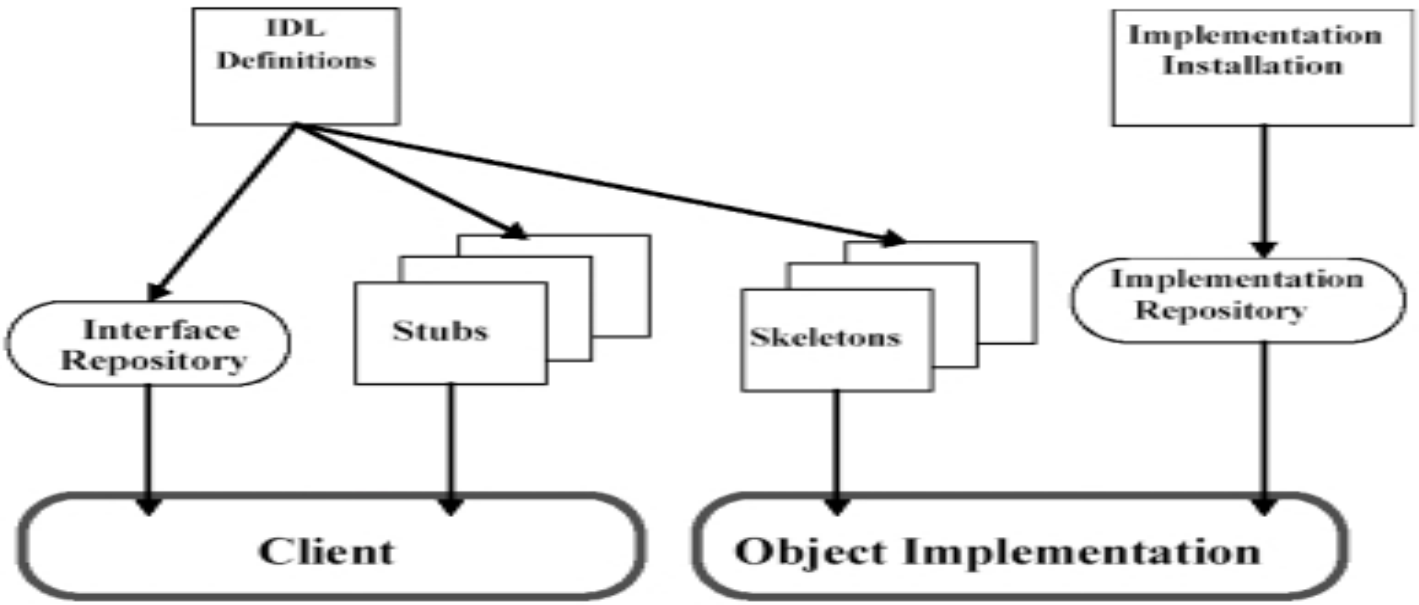
Interface Definition Language (IDL)

-IDL provides a programming language neutral way to define how a service is implemented.

÷It is an intermediary language between specification languages such as the UML and programming languages such as C, C++ etc.

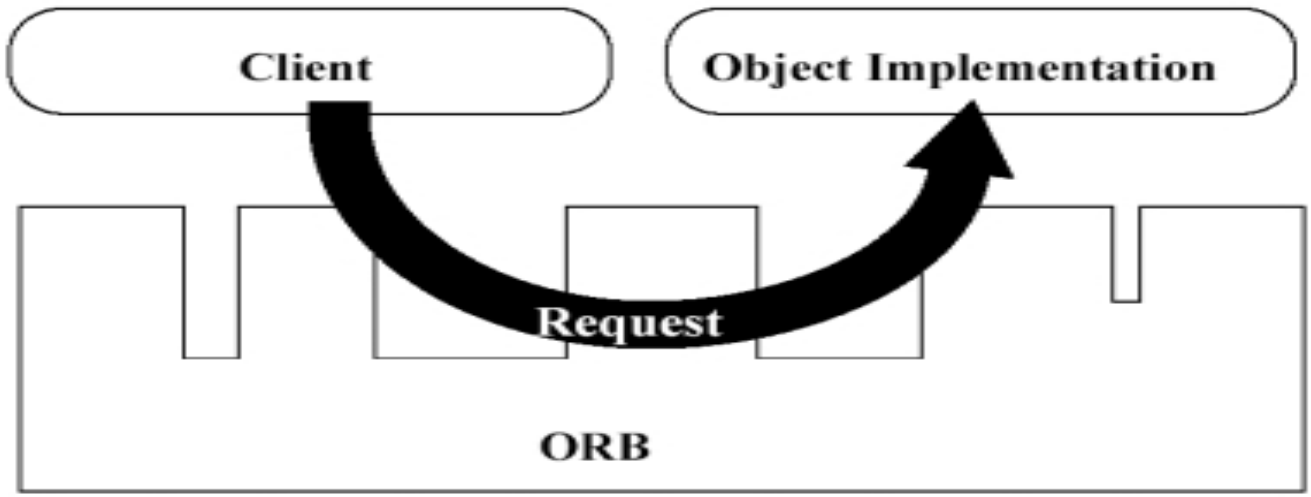
÷It provides an abstract representation of the interfaces that a client will use and a server will implement.

÷Clients and object implementation are then isolated by three mechanisms: an IDL stub on the client end, an ORB, and a corresponding skeleton on the object implementation end.



Object Request Broker

-The interface the client sees is completely independent of where the object is located, what programming language it is implemented in, or any other aspect that is not reflected in the object's interface.



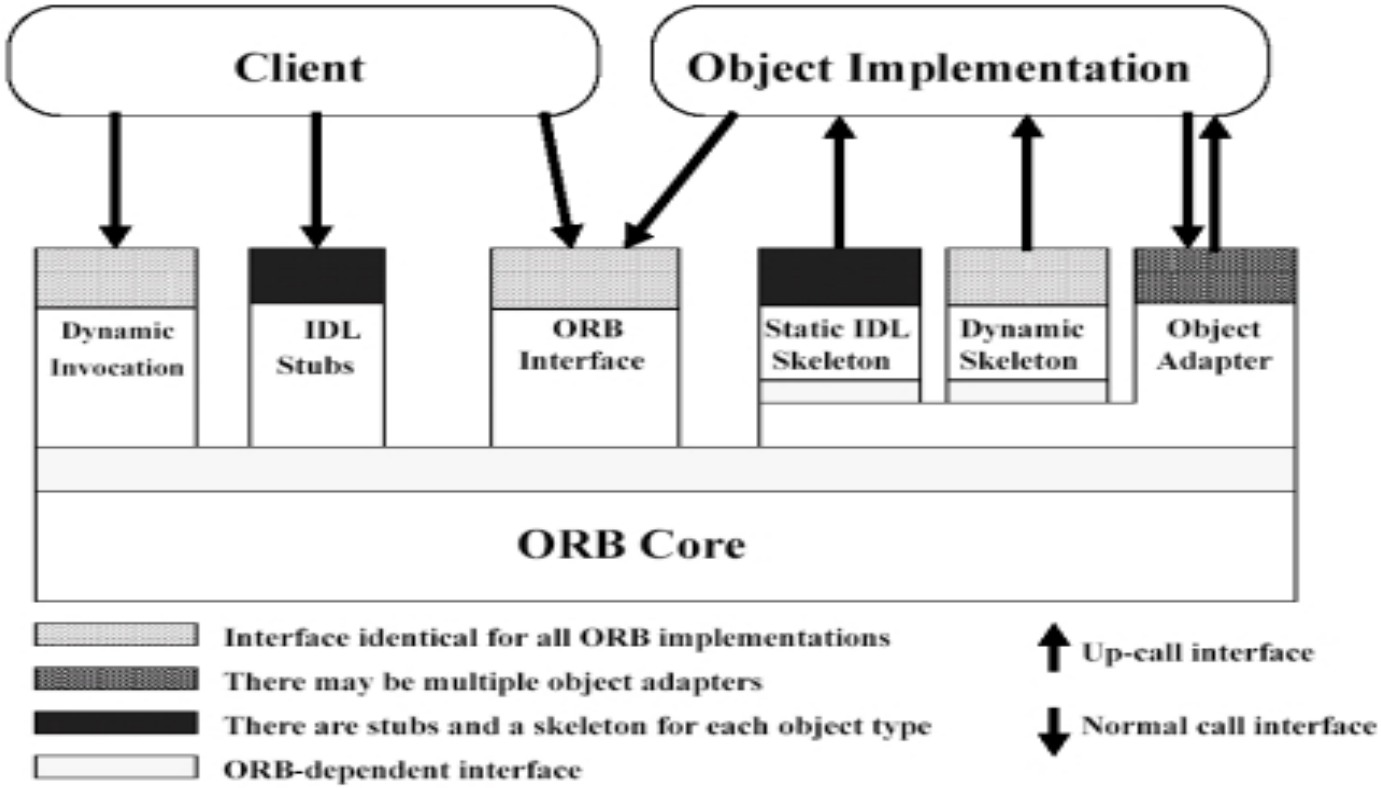
-The ORB is responsible for:

÷finding the object implementation for the request,

÷preparing the object implementation to receive the request,

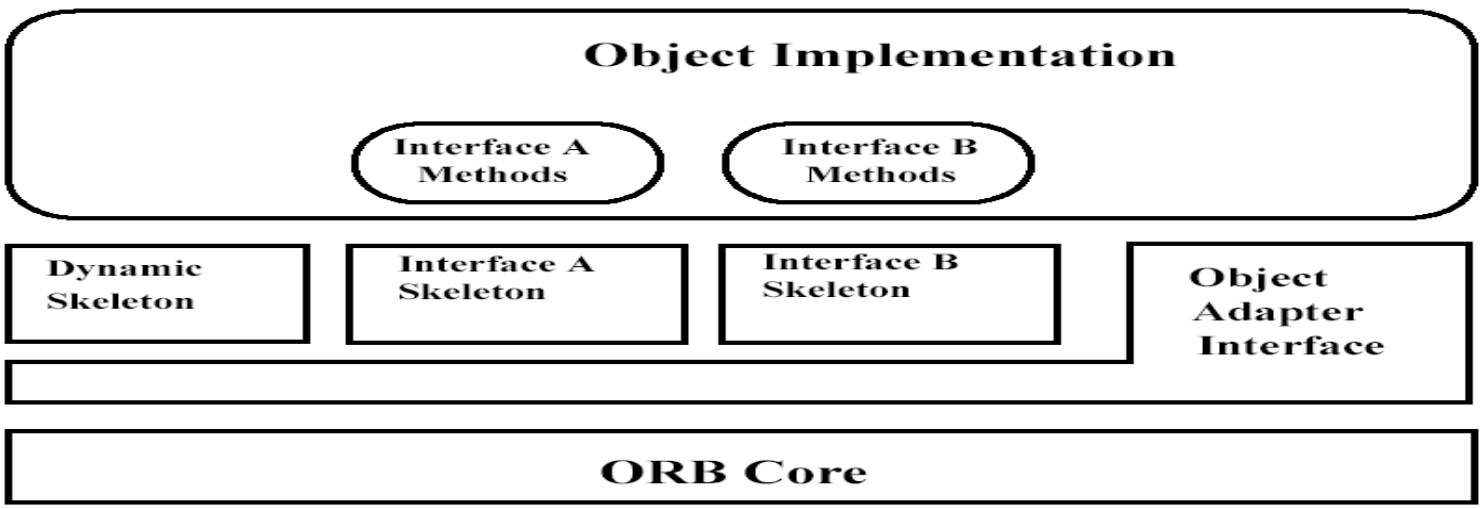
÷communicating the data making up the request.

Structure of Object Request Interfaces



Object Adapter

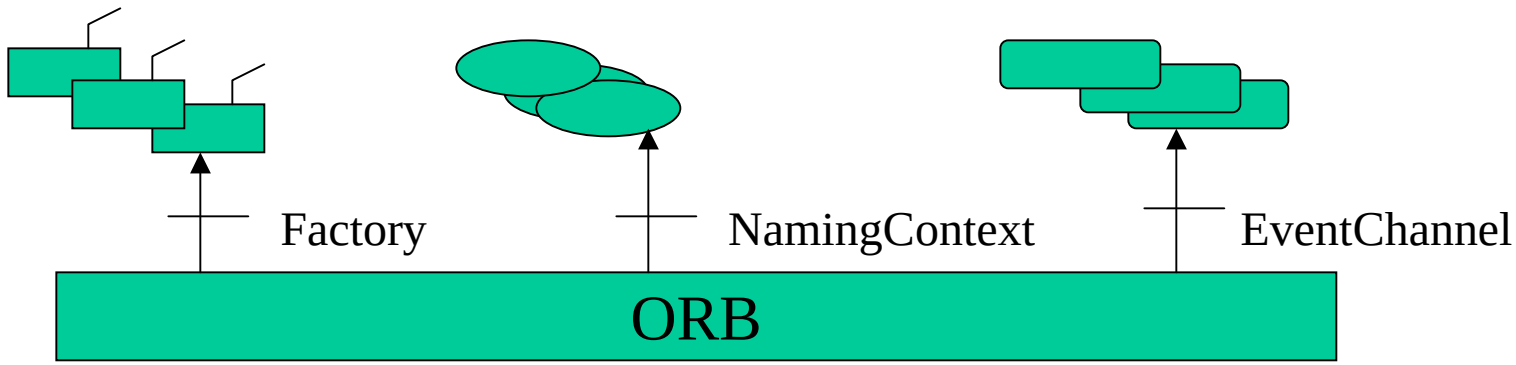
- An object adapter is the primary means for an object implementation to access ORB services such as object reference generation.
- Object adapters are responsible for the following functions:
  - –Generation and interpretation of object references
  - –Method invocation
  - –Security of interactions
  - –Object and implementation activation and deactivation
  - –Mapping object references to the corresponding object implementations
  - –Registration of implementations



Common Object Services (COS)

-Up to 15 services are currently available that assist the ORB.

÷They are defined on top of the ORB, as standard CORBA objects with IDL interfaces



-Popular services include:

•*Naming Service*: provides a way for CORBA clients (and servers) to find objects on the network.

•*Event Service*: stores and delivers events sent by clients or servers to their target objects.

•*Security Service*: provides a means to authenticate messages, authorize access to objects, and provide secure communications.

•*Transaction Service*: defines a means to control an action against a database or other subsystem.

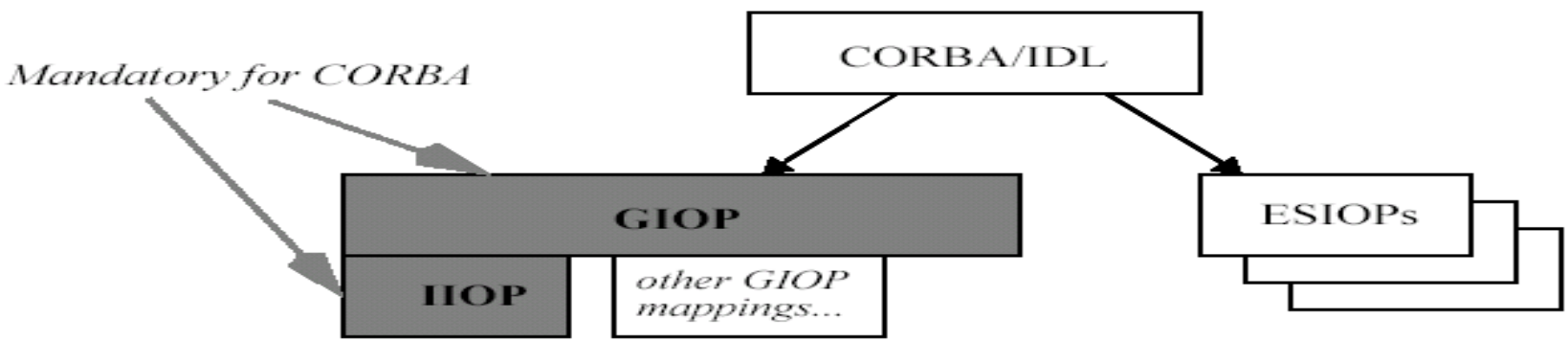
ORB Interoperability Architecture

-One of the goals of the CORBA specification is that client and object implementations are portable.

-Interoperability is more important in a distributed system than portability.

-CORBA 2.0 added interoperability as a goal in the specification, under the form of interoperability protocols:

- ÷Inter-ORB Bridge Support
- ÷General Inter-ORB Protocol (GIOP)
- ÷Internet Inter-ORB Protocol (IIOP)
- ÷Environment-Specific Inter-ORB Protocols (ESIOPs)



Inter-ORB Bridge Support

-The role of a bridge is to ensure that content and semantics are mapped from the form appropriate to one ORB to that of another, so that users of any given ORB only see their appropriate content and semantics.

-The General Inter-ORB Protocol (GIOP) specifies a standard transfer syntax (low-level data representation) and a set of message formats for communications between ORBs.

÷The protocol is specifically built for ORB to ORB interactions and is designed to work directly over any connection-oriented transport protocol that meets a minimal set of assumptions.

-**IIOP (Internet Inter-ORB Protocol)** is a TCP/IP implementation of GIOP.

÷IIOP is used in other systems that do not even attempt to provide the CORBA API (e.g, “RMI over IIOP”, EJB etc.)

÷Because they all use IIOP, programs written to these APIs can interoperate with each other and with CORBA programs.

Examples of CORBA Products

ORB	***Description***
Orbacus	A popular Java and C++ ORB from Iona Technologies
WebSphere	A popular application server with an ORB from IBM
Netscape Communicator	Netscape browsers have a version of VisiBroker embedded in them
Sun Java 2 Platform	Provides an ORB and two CORBA programming models: -RMI programming model -IDL programming model

Examples ORBs implementations

- Client- and Implementation-resident ORB
- Server-based ORB
- System-based ORB
- Library-based ORB