CORBA
CHAPTER 5 Introduction to CORBA

- Distributed objects in CORBA,
- CORBA components,
- architectural features,
- method invocations,
- static and dynamic: IDL (Interface Definition Language) models and interfaces.
- Structure of CORBA IDL,
- CORBA's self-describing data;
- CORBA interface repository.
- Building an application using CORBA
Distributed Object System

• Objects are Distributed but behave like a local Object..

• To communicate Message Passing is used.

• Middleware promotes transparency in replication and consistency management, Object location and communication.

• **Advantages**
  – Rapid Application Development.
  – Reduction in time and cost of the Development.
  – Higher degree of system integration.
  – Support reusable and pluggable business component.
Why We Need CORBA?

Need a solution to develop, deploy, and integrate systems in a distributed heterogeneous environment.

- Diverse OS – Unix, Windows, MacOS etc.
- Diverse Network – TCP/IP, Ethernet, ATM, etc.
- Diverse Programming Language – applications programmed in C++, JAVA, COBOL etc.
- Diverse Hardware Platform.
- Coexist with legacy systems.
WHAT IS CORBA?

- Architecture for interoperable distributed computing
- Developed by the **OMG** (Object Management Group)
  
  A consortium of over 700 companies.
  
  **Goal**
  
  To develop, adopt, and promote standards for the development and deployment of applications in distributed heterogeneous environments.

**CORBA – Common Object Request Broker Architecture**

Middleware that provides the necessary framework and API for developing distributed applications

**Middleware** :- Middleware is the software that sits ‘in the middle' between applications working on different operating systems
CORBA

is a standard *Middleware Technology* for the communication of *heterogeneous Distributed Objects* of a distributed application.

Why Distributed Applications?

**Data is distributed**
Administrative and ownership reasons, Heterogeneous systems, Shared by multiple applications, Scalability.

**Computation is distributed**
Scalability: multiprocessing, Take computation to data, Heterogeneous architectures.

**Users are distributed**
Multiple users interacting and communicating via distributed applications
Common Object Request Broker Architecture
Overview of CORBA

OMG Reference Model of CORBA
Overview of CORBA

Reference: Object Oriented Concepts, Inc. CORBA and OmniBroker

Compiled By BJ
Main Components of CORBA

• 1) ORB Core

• 2) OMG Interface Definition Language (OMG IDL) & CORBA Objects

• 3) Interface Repository (IFR)

• 4) Stubs and Skeletons

• 5) IDL Compiler

• 6) Dynamic Invocation Interface, Dynamic Skeleton Interface

• 7) Object Adapter (OA)

• 8) Inter-ORB Protocols

• 9) Orb interface & Implementation Repository (IMR)
Components of CORBA
Object Request Broker (ORB)

- ORB provides communication and activation Infrastructure for distributed object application.
- The key feature of ORB is transparency of how it facilitates client-Object communication. It provide following transparencies

  **Key Feature of the ORB are**

  **Transparent Object Location** – Clients do not know where target objects reside.

  **Transparent Object Implementation** – Clients do not know how objects are implemented. What programming language, or what OS it is running under.

  **Transparent Object Execution State** – Clients do not know if objects are activated or deactivated when making requests. ORB takes care of that.

  **Transparent Object Communication Mechanism** – Clients do not know what mechanism ORB uses to transfer requests, i.e. local method call etc.
Object Request Broker (ORB)

- Based on client/server model of distributed computing, CORBA inserts a "broker" between client and server.

- Broker, distributed service, provides basic messaging and communication between client and server.

- Hides issues such as distribution and heterogeneity.

- Implemented by ORB Vendor as API libraries linked with Client and ServerApp.

- Intercepts calls, locates remote object, passes parameters, returns results or error messages, REGARDLESS OF THE OBJECTS LOCATION, ITS PROGRAMMING LANGUAGE OR EVEN THE OPERATING SYSTEMS INVOLVED!!
Object Request Broker (ORB)

- ORB is a software component that mediates the transfer of messages from a program to an object located on a remote network host.
• Client and Server ORB’s communicate over a network.

• It implements location transparency. Exactly same request mechanism is used by client regardless of server object location.

• Distributed objects are identified by object references, which are typed by IDL interfaces.

• Manipulates Object references. That is Marshals and Unmarshals them.

• It provides the means to obtain an Initial reference to an object implementing CORBA service.
ORB Core

Main Function: To deliver requests to objects and return any responses to the clients making the requests.

- Client
- Object Implementation
- Request
- ORB
CORBA Objects

• **CORBA objects** are abstract objects in a CORBA system that provide distributed object capability between applications in a network.
Object Model

- The general organization of a CORBA system.
ORB Technology

- Provide illusion of locality.
- Transparency.
- Provides Interface Definitions, Location and activation.
OMG IDL (Interface Definition Language)

An interface is a description of the operations that are offered by an object.

- OMG IDL is a declarative language for defining the interfaces of CORBA objects. All CORBA Objects must be described in OMG IDL.
- The IDL file is then compiled using the special IDL compiler to the specified programming language.
- Currently there are IDL mappings to many popular programming language including JAVA, C, and C++.
- This interface is then implemented using the programming language specified.
- Due to objects having this common interface, inter-object communication is possible.
CORBA IDL

- **OMG/ISO IDL (Interface Definition Language)**
  - Separates the Interface from the Implementation
    - multiple-inheritance, strongly typed, public interface specification language;
    - independent of any particular language/compiler;
    - mappings will be provided for many languages/compilers;
    - *not* a programming language.

- **Enables Interoperability**
CORBA IDL

Written by the Software Developer.

– ORB, CORBA Services, CORBA Facilities can interoperate.

– Language neutral way to define, how a service can be implemented.

– Using IDL Compiler they are compiled for a specific O.S. and Programming languages
  – Allows Client and Server applications to be written in different (several) languages.

– A “contract” between Clients and Servers
  • Both MUST have the exact same IDL
  • Specifies messages and data that can be sent by Clients and received by Servants

Compiled By BJ
## IDL – Java Mapping

<table>
<thead>
<tr>
<th>IDL Constructs</th>
<th>Java Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>Package</td>
</tr>
<tr>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>Char</td>
<td>char</td>
</tr>
<tr>
<td>Octet</td>
<td>byte</td>
</tr>
<tr>
<td>String</td>
<td>Java.lang.string</td>
</tr>
<tr>
<td>Short</td>
<td>Short</td>
</tr>
<tr>
<td>Long</td>
<td>Int</td>
</tr>
<tr>
<td>Long long</td>
<td>Long</td>
</tr>
<tr>
<td>Float</td>
<td>Float</td>
</tr>
<tr>
<td>Enum</td>
<td>A Java class with a static final int member for each enum member.</td>
</tr>
<tr>
<td>Struct</td>
<td>A Java Class. All methods and instance variables are public.</td>
</tr>
<tr>
<td>interface</td>
<td>A Java interface. Compiler also generates a stub and skeleton code to implement location transparency.</td>
</tr>
<tr>
<td>exception</td>
<td>A Java class that extends omg.CORBA.UserException class.</td>
</tr>
</tbody>
</table>
CORBA IDL
Stubs and Skeletons

The IDL compiler will generate a client side stub and server side skeleton code. Handles the lower network management in distributed applications.

When client invoke an IDL-defined operation on an object reference as if it were a local method, it must link in the stub code.

- Stub is a mechanism that creates and issues request on behalf of the client.
- Stub handles the marshalling of a request, that is converting data structure into wire format.

Once a request reaches the server side, the skeleton code is used to invoke the right method on the right implementation.

- Skeleton code translates wire format data into memory data format (unmarshaling).
Overview of CORBA

Reference: Object Oriented Concepts, Inc. CORBA and OmniBroker

Compiled By BJ
Dynamic Invocation Interface (DII)
Dynamic Skeleton Interface (DSI)

The static stub and skeleton codes are limited in use.

- **DII** defines a *request* so that clients who know the
  • object reference
  • interface type
- Can build requests without having to rely on the IDL generated stub code.

- **DSI** deals with requests sent by clients in a generic manner.

- Looks at the requested operation and its arguments and interpreting the semantics dynamically.
Dynamic invocation interface

- Invoking operations can be done through either static or dynamic interfaces.
- Static invocation interfaces are determined at compile time, and they are presented to the client using stubs.
- The DII, on the other hand, allows client applications to use server objects without knowing the type of those objects at compile time. It allows a client to obtain an instance of a CORBA object and make invocations on that object by dynamically constructing requests.
- DII uses the interface repository to validate and retrieve the signature of the operation on which a request is made.
- CORBA supports both the dynamic and the static invocation interfaces.
Dynamic skeleton interface

• Analogous to the DII is the server-side dynamic skeleton interface (DSI), which allows servers to be written without having skeletons, or compile-time knowledge, for the objects being implemented.

• Unlike DII, which was part of the initial CORBA specification, DSI was introduced in CORBA 2.0. Its main purpose is to support the implementation of gateways between ORBs which utilize different communication protocols.

• However, DSI has many other applications beside interoperability. These applications include interactive software tools based on interpreters and distributed debuggers.
Interface Repository

- Provides a dynamic mechanism for CORBA-based applications to access OMG IDL type system when it is executing.
- Applications must know the types of values to be passed as request arguments, also applications must know the types of interfaces supported by the objects being used.
- Most applications know these values statically, because of pre-compiled code. It is necessary sometimes to access these information at runtime
- The IR can be thought of as a set of objects that encapsulate the IDL definitions of all CORBA types available in a particular domain.

- The IR provides another way to specify the interfaces to objects. Interfaces can be added to the interface repository service. Using the IR, a client should be able to locate an object that is unknown at compile time, find information about its interface, then build a request to be forwarded through the ORB.
ORB Interoperability

Common Protocols to facilitate transmission of CORBA requests and replies between ORBs.

**GIOP – General Inter-ORB Protocol**
- Specified transfer syntax and a standard set of message formats for ORB interoperation over any connection-oriented transport.

**IIOP – Internet Inter-ORB Protocol**
- Becoming standard for most ORB products
- An implementation of the GIOP for TCP/IP

**ESIOP – Environment Specific Inter-ORB Protocol**
- Specific protocol made for already existing distribute computing infrastructure.
- To bridge to other ORBs, use IIOP.
**ORB Interoperability**

In addition to standard protocols, standard object reference formats are also necessary for ORB interoperability.

ORBs use the contents of object references to help determine how to direct requests to objects.

**IOR** – Interoperable Object Reference

- Stores information needed to locate and communicate with an object over one or more protocols.
- For example, an IOR containing IIOP information specifically stores hostname and TCP/IP port number information.
Object Adapter

Serves as the glue between CORBA Object implementation and the ORB itself.

Main functionalities of the Object Adapter:
- **Request Demultiplexing** – request arrives for an object, make sure the right servant is located.
- **Operation Dispatching** – once the servant is located, pass along the request to the servant.
- **Activation and deactivation** – activate the object, and deactivate the object when no longer needed.
- **Generating Object References** – generate references to objects for clients to use

An object adapter is the primary way that an object implementation accesses services provided by the ORB. Such services include object reference generation and interpretation, method invocation, security of interactions, and object and implementation activation and deactivation.
Object Adapter

- Object adapters mediate between CORBA objects and programming language implementations (servants). Object adapters provide a number of services, including the creation of CORBA objects and their references, dispatching requests to the Appropriate servant that provides an implementation for the target object, and Activation and deactivation of CORBA objects.
Portable Object Adapter

– A central CORBA goal: Programs using different ORB’s (provided by different ORB Vendors) can still communicate.

– In the early days of CORBA, this was not possible
– The POA was adopted as the solution
– Can be viewed as an API/Framework
  • Set of classes and method
– Sits between ORB’s and Servants

– Job is to:
  • Receive messages from ORB’s
  • Activate the appropriate Servant
  • Deliver the message to the Servant
Portable Object Adapter – (POA)

- **Object Adapter**: Activation policy for a group of objects.

- **Wrappers**: It adapts the program in such a way that clients can see the program as an Object.

- **POA**: Component that is responsible for making the server side code appear as CORBA objects to clients.

- It supports portability across different ORBs.

- The Java CORBA ORB shipped in J2SE is multi-threaded. On the server side, we have a thread pool such that each incoming request is handled by a separate thread.
CORBA 3.0 Supports Portable Object Adapter

*Object adapter* is the mechanism that connects a request using an object reference with the proper code to service that request. The POA mediates the CORBA objects and the programming implementations. The main purpose of POA is to provide portability for CORBA server applications. Portability from the implementation of the ORB.

Portable Object Adapter (POA) allows:

- Creation of CORBA Objects
- Demultiplexing of requests made on each CORBA object
- Dispatching requests to the appropriate implementation that the CORBA Object represents
- Activation/Deactivation of CORBA Objects
1. Portable Object Adapter (POA)

Server Application

ORB

Request

Compiled By BJ
CORBA 3.0

1. Portable Object Adapter (POA)
CORBA 3.0

1. Portable Object Adapter (POA)
2) CORBA Messaging

- The CORBA messaging fills a big hole in the consistency of the CORBA specification
- It helps reliably communicating over unreliable connections.
- The Client / Servers are not that tightly coupled anymore as in the synchronous invocation.
- Makes CORBA usable in large distributed systems.
- Asynchronous messaging
- Time-independent invocations
- Facilities for specifying messaging quality of service (QoS)
Thank You