The Common Object Request Broker Architecture (CORBA)

CORBA

- CORBA is a standard architecture for distributed objects systems
- CORBA is designed to allow distributed objects to interoperate in a heterogenous environment, where objects can be implemented in different programming languages and/or deployed on different platforms
CORBA vs. Java RMI

- RMI is a proprietary facility and supports objects written in the Java programming language only.
- CORBA is an architecture that was developed by the Object Management Group (OMG), an industrial consortium.

CORBA

- CORBA is a very rich set of protocols.
- A distributed object facility which adhere to these protocols is said to be CORBA-compliant.
- The distributed objects the facility supports can interoperate with objects supported by other CORBA-compliant facilities.
The basic architecture

CORBA object interface

- A distributed object is defined using an interface similar to the remote interface file in Java RMI
- Universal language with a distinct syntax, known as the **CORBA Interface Definition Language (IDL)**
- For many languages there is a standardized mapping from CORBA IDL
Cross-language CORBA application

To allow ORBs to be interoperable, the OMG specified a protocol known as the General Inter-ORB Protocol (GIOP), a specification which “provides a general framework for protocols to be built on top of specific transport layers”

Inter-ORB Protocol (IIOP) = GIOP applied to the TCP/IP transport layer
Inter-ORB protocols

The IIOP specification includes the following elements:

- **Transport management requirements**
  - connection and disconnection requirements
  - roles for object client and object server in making and unmaking connections

- **Definition of common data representation**
  - a coding scheme for marshalling and unmarshalling data of each IDL data type

- **Message formats**

Object bus

- An ORB which adheres to the specifications of the IIOP may interoperate with any other IIOP-compliant ORBs over the Internet
- **“Object bus”,** where the Internet is seen as a bus that interconnects CORBA objects
CORBA object references

A CORBA object reference is an abstract entity mapped to a language-specific object reference by an ORB, in a representation chosen by the developer of the ORB.

For interoperability, OMG specifies a protocol for the abstract CORBA object reference object, known as the Interoperable Object Reference (IOR) protocol.

Interoperable Object Reference (IOR)

An IOR is a string that contains encoding for the following information:

- The type of the object
- The host where the object can be found
- The port number of the server for that object
- An object key, a string of bytes identifying the object, used by an object server to locate the object
CORBA Naming Service

- CORBA specifies a generic directory service. The **Naming Service** serves as a directory for CORBA objects.
- The Naming Service allows names to be associated with object references.

To export a distributed object, a CORBA object server contacts a Naming Service to **bind** a symbolic name to the object.
- The Naming Service maintains a database of names and the objects associated with them.
- The Naming Service **resolves** an object name returning a reference to the object.
- The API for the Naming Service is specified in interfaces defined in IDL.
CORBA Naming Service

- The CORBA object naming scheme is necessarily complex.
- Since the name space is universal, a standard naming hierarchy is defined.

![Diagram showing a hierarchy of naming contexts with object names]

CORBA Naming Service

- The full name of an object, including all the associated naming contexts, is known as a compound name.

```
<naming context> ...<naming context><object name>
```

- Naming contexts and name bindings are created using methods provided in the Naming Service interface.
Interoperable Naming Service

- The *Interoperable Naming Service (INS)* is a URL-based naming system based on the CORBA Naming Service.
- It allows applications to share a common initial naming context and provide a URL to access a CORBA object.

CORBA Object Services

CORBA specifies services commonly needed in distributed applications:
- Naming Service
- Concurrency Service
- Event Service
- Logging Service
- Scheduling Service
- Security Service
- Trading Service: for locating a service by the type (instead of by name)
- Time Service: a service for time-related events
- Notification Service
- Object Transaction Service
**Object Adapters**

<table>
<thead>
<tr>
<th>distributed object implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>object adapter</td>
</tr>
<tr>
<td>ORB</td>
</tr>
</tbody>
</table>

**Object Adapter**

- An object adapter assists an ORB in delivering a client request to an object implementation.
- When an ORB receives a client’s request, it locates the object adapter associated with the object and forwards the request to the adapter.
- The adapter interacts with the object implementation’s skeleton, which performs data marshalling and invokes the appropriate method in the object.
The Portable Object Adapter

- There are different types of CORBA object adapters.
- The **Portable Object Adapter**, or **POA**, is a particular type of object adapter that is defined by the CORBA specification.
- An object adapter that is a POA allows an object implementation to function with different ORBs.

The Java IDL
Java IDL – Java’s CORBA facility

- IDL is part of the Java 2 Platform
- The Java IDL facility includes a CORBA Object Request Broker (ORB), an IDL-to-Java compiler, and a subset of CORBA standard services
- Java also provides a number of CORBA-compliant facilities, including RMI over IIOP, which allows a CORBA application to be written using the RMI syntax and semantics

Key Java IDL packages

- org.omg.CORBA – contains interfaces and classes providing the mapping of the OMG CORBA APIs to the Java programming language
- org.omg.CosNaming - contains interfaces and classes providing the naming service for Java IDL
Java IDL tools

Java IDL provides a set of tools needed for developing a CORBA application:

- idlj - the IDL-to-Java compiler
- orbd - a server process which provides Naming Service and other services
- servertool – provides a command-line interface for application programmers to register/unregister an object, and startup/shutdown a server

The CORBA interface

```
module HelloApp
{
    interface Hello
    {
        string sayHello();
        oneway void shutdown();
    }
};
```
Compiling the IDL file

- The IDL is compiled as follows:
  
  `idlj -fall Hello.idl`

- The `-fall` command option is necessary for the compiler to generate all the files needed.

- If the compilation is successful, the following files can be found in a `HelloApp` subdirectory:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HelloOperations.java</td>
<td>Java operations interface</td>
</tr>
<tr>
<td>HelloHelper.java</td>
<td></td>
</tr>
<tr>
<td>_HelloStub.java</td>
<td></td>
</tr>
<tr>
<td>Hello.java</td>
<td></td>
</tr>
<tr>
<td>HelloHolder.java</td>
<td></td>
</tr>
<tr>
<td>HelloPOA.java</td>
<td></td>
</tr>
</tbody>
</table>

HelloOperations.java

- The file `HelloOperations.java` is the `Java operations interface`.

- It is a Java interface file that is equivalent to the CORBA IDL interface file (`Hello.idl`).

- You should look at this file to make sure that the method signatures correspond to what you expect.
Hello.java

- The signature interface file combines the characteristics of the Java operations interface (HelloOperations.java) with the characteristics of the CORBA classes that it extends (org.omg.CORBA.Object, org.omg.CORBA.portable.IDLEntity).

HelloHelper.java

- The Java class HelloHelper provides auxiliary functionality needed to support a CORBA object in the context of the Java language.
- In particular, a method, narrow, allows a CORBA object reference to be cast to its corresponding type in Java, so that a CORBA object may be operated on using syntax for Java object.
The Java class `_HelloStub` is the stub file, which interfaces with the client object. It extends `org.omg.CORBA.portable.ObjectImpl` and implements the `Hello.java` interface.

The Java class `HelloImplPOA` is the skeleton combined with the portable object adapter.
Server-side classes

- On the server side, two classes need to be provided
  - The servant, `HelloImpl`, is the implementation of the `Hello` IDL interface
  - The object server, `HelloServer`

---

The servant

```java
import org.omg.CosNaming.*;
import org.omg.CORBA.ORB;

class HelloImpl extends HelloPOA
{
  private ORB orb;

  public void setORB(ORB _orb)
  { orb = _orb; }

  public String sayHello()
  { return "Hello world !! "; }

  public void shutdown()
  { orb.shutdown(false); }
}
```
import org.omg.CosNaming.*;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.*;

public class HelloServer
{
   public static void main(String args[])
   {
      try
      {
         ORB orb = ORB.init(args, null);
         POA rootpoa = (POA)orb.resolve_initial_references("RootPOA");
         rootpoa.the_POAManager().activate();
         HelloImpl helloImpl = new HelloImpl();
         helloImpl.setORB(orb);
         org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
         Hello href = HelloHelper.narrow(ref);
         [...]  
      }
      catch(Exception e)
      {  System.out.println(e);
      }
   }
} // class

org.omg.CORBA.Object objRef =
   orb.resolve_initial_references("NameService");
NamingContextExt ncRef =
   NamingContextExtHelper.narrow(objRef);
String name = "Hello";
NameComponent path[] = ncRef.to_name( name );
ncRef.rebind(path, href);
System.out.println("HelloServer ready
   and waiting ...");
orb.run();
}
catch(Exception e)
{  System.out.println(e);
}
} // main
} // class
The object client /1

- The client code is responsible for:
  - creating and initializing the ORB
  - looking up the object using the Interoperable Naming Service
  - invoking the narrow method of the Helper object to cast the object reference to a reference to a Hello object implementation
  - invoking remote methods using the reference
- The object’s sayHello method is invoked to receive a string, and the object’s shutdown method is invoked to deactivate the service.

The object client /2

```java
import org.omg.CosNaming.*;
import org.omg.CORBA.ORB;

public class HelloClient
{ static Hello helloImpl;
  public static void main(String args[]) { try {
    ORB orb = ORB.init(args, null);
    org.omg.CORBA.Object objRef =
    orb.resolve_initial_references(
      "NameService");
    NamingContextExt ncRef =
      NamingContextExtHelper.narrow(
        objRef);
    helloImpl = HelloHelper.narrow(
      ncRef.resolve_str("Hello");
    [...]
```

The object client /3

```java
System.out.println(
    helloImpl.sayHello());
helloImpl.shutdown();
}
catch(Exception e)
{
    System.out.println(e);
}
}
```

Starting the Java ORB on the server

The Java ORB daemon `orbd` includes a Naming Service

```
orbd -ORBInitialPort 1050
-ORBInitialHost servermachinename
```
Running the application

On the server

```java
java HelloServer
   -ORBInitialHost nameserverhost
   -ORBInitialPort 1050
```

On the client

```java
java HelloClient
   -ORBInitialHost nameserverhost
   -ORBInitialPort 1050
```

N.B.: `nameserverhost` is the host on which the IDL name server is running