Chapters 4 and 10: COMMUNICATION
Part 2c

Communications in Distributed Systems

Distributed Objects, RMI, **CORBA**

Distributed Computing,
M. L. Liu

Thanks to the authors of the textbook [**MLL**] for providing the base slides. I made several changes/additions.
These slides may incorporate materials kindly provided by Prof. Dakai Zhu.
So I would like to thank him, too.

Turgay Korkmaz
korkmaz@cs.utsa.edu

Chapters 4 and 10: Communications

- FUNDAMENTALS
  - Layered Protocols
  - Types of communications
- REMOTE PROCEDURE CALL
  - Basic RPC Operation
  - Parameter Passing
  - RPC operation
  - RPC Examples
  - Asynchronous RPC
  - RMI (some from chapter 10, but most from web)
  - CORBA (some from chapter 10, but most from web and [MLL])
- MESSAGE-ORIENTED COMMUNICATION
  - Transient and Persistent Communication
- STREAM-ORIENTED COMMUNICATION
  - Support for Continuous Media and Quality of Service
  - Stream Synchronization
- MULTICAST COMMUNICATION
  - Application-Level Multicasting
  - Gossip-Based Data Dissemination

Objectives

- To understand how processes communicate (the heart of distributed systems)
- To understand computer networks and their layers
- To understand client-server paradigm and low-level message passing using **sockets**
- To learn higher-level communication mechanisms
  - RPC, RMI, CORBA
- To understand various forms of communications and their issues
  - Msg-, Stream-oriented communication, multicast, etc.

CORBA

Platform and language independent RMI

- The Common Object Request Broker Architecture (CORBA) is a standard architecture for a distributed objects system.
- CORBA is designed to allow distributed objects to interoperate in a heterogeneous environment, where objects can be implemented in different programming language and/or deployed on different platforms
- Java RMI is platform independent too but it is language dependent

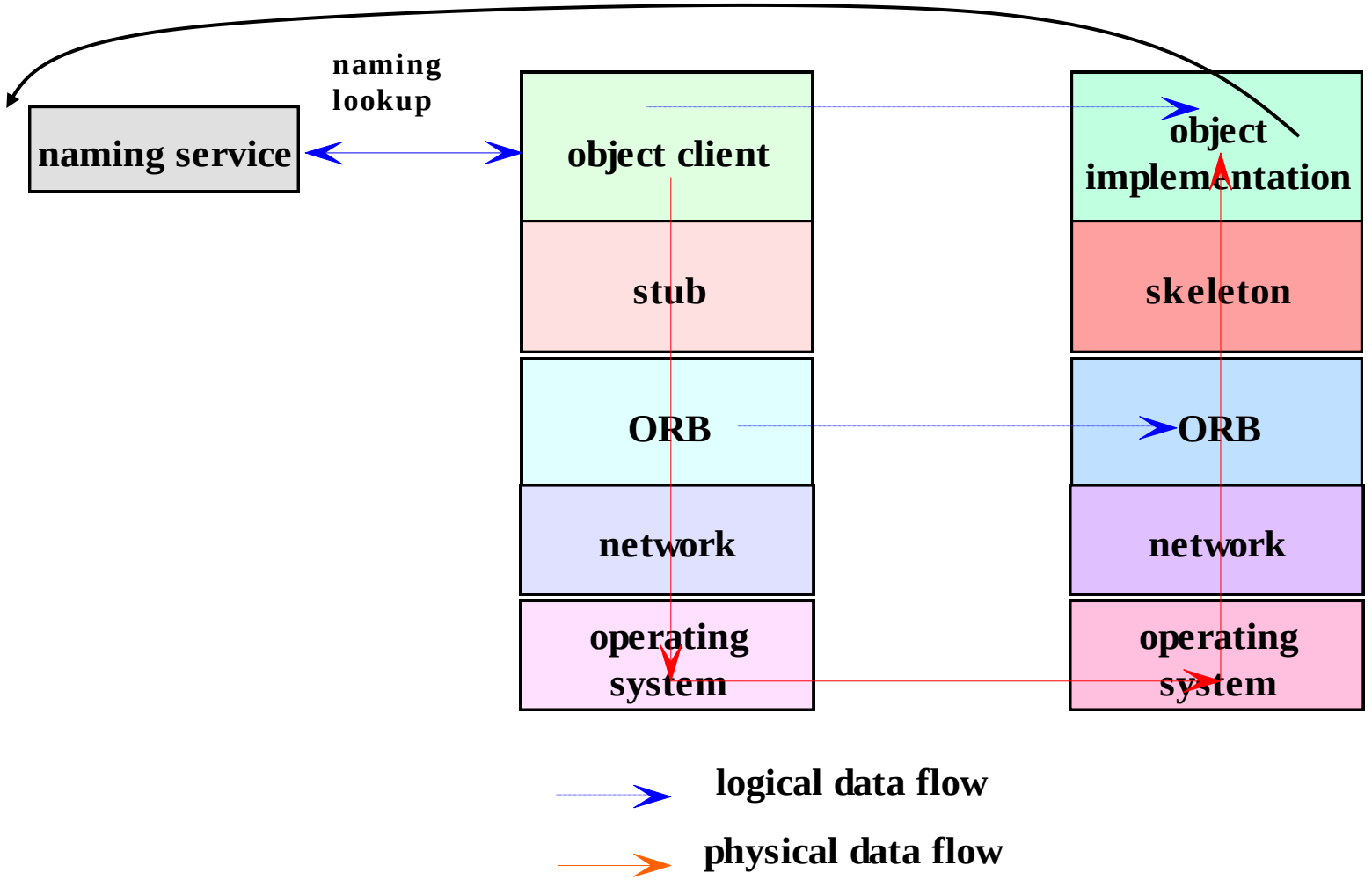
CORBA vs. Java RMI

- CORBA differs from the architecture of Java RMI in one significant aspect:
  - RMI is a proprietary facility developed by Sun MicroSystems, Inc., and supports objects written in the Java programming language only.
  - CORBA is a suite of specifications developed by the Object Management Group (OMG), http://www.omg.org/
    - Using a facility supporting CORBA, objects can be written in any language

CORBA

- CORBA is not itself a distributed objects facility; instead, **it is a set of protocols**.
- A distributed object facility which adhere to these protocols is said to be CORBA-compliant, and the distributed objects that the facility support can interoperate with objects supported by other CORBA-compliant facilities.
- CORBA is a very rich set of protocols. But we will focus on the key concepts of CORBA related to the distributed objects paradigm.
- We will also study a facility based on CORBA: the Java IDL.

The Basic Architecture



CORBA Object Interface

- Since CORBA is language independent, the interface is defined using a universal language with a distinct syntax, known as the CORBA Interface Definition Language (IDL).
- The syntax of CORBA IDL is similar to Java, C++
  - Object defined in a CORBA IDL file can be implemented in a large number of diverse programming languages, e.g., C/C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLScript.
- For each language, OMG has a standardized mapping from CORBA IDL to the language,
  - So a compiler can be used to process a CORBA interface to generate the proxy files needed to interface with an object implementation or an object client written in any of the CORBA-compatible languages.

CORBA IDL

- A distributed object is defined using a software file similar to the remote interface file in Java RMI.
- Used to define module, interface, types, attributes and method signatures

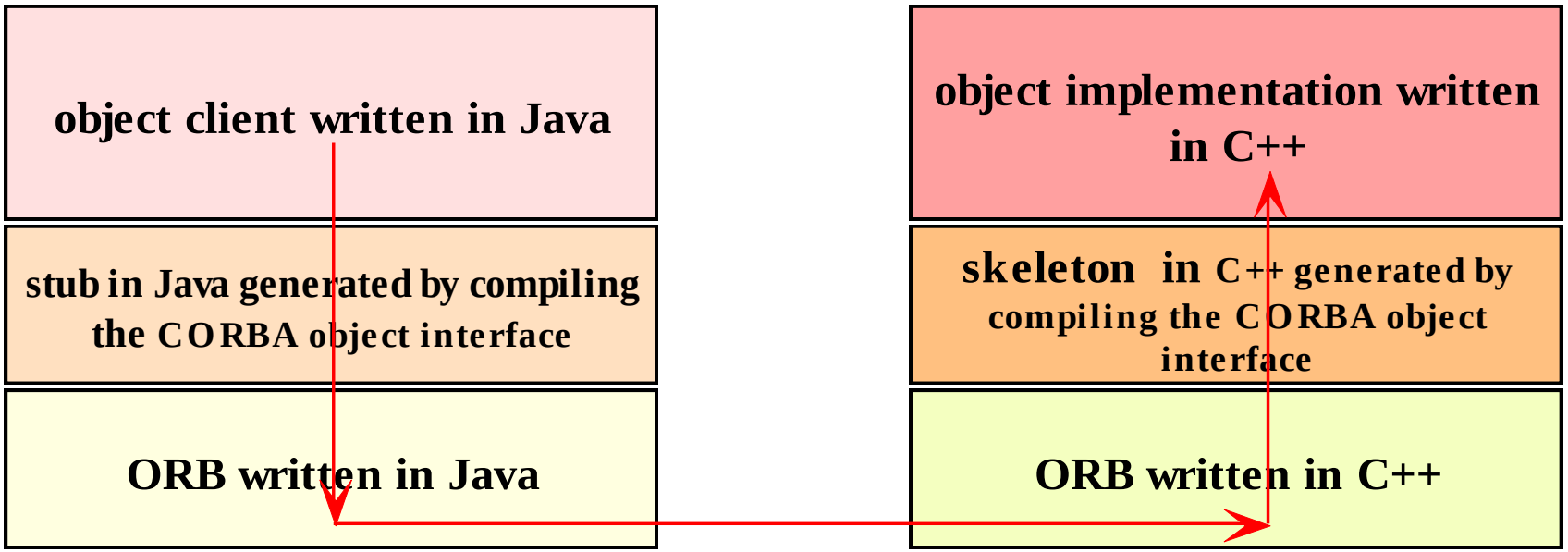
```
//Hello.idl
module HelloApp {
   interface Hello {
      string add(in string s);
      string say();
   };
};
```

  - Module → name space, Java package
  - Interface → define a set of methods for CORBA objects that implement the interface
  - Methods → parameters and results
  - Types → primitive or constructed using typedefs
  - Attributes → private to CORBA objects
  - Inheritance → multi-inheritance

CORBA IDL (cont'd)

- Parameters and results about methods
  - Parameters are specified by keywords: **in, out**, or **inout**
  - Primitive type or constructed type → pass by value
  - CORBA object of interface type → remote object reference
- User-defined exceptions in interfaces and thrown by methods in the interface
- At-most-once invocation semantics by default

Cross-language CORBA application



Inter-ORB Protocols

- 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."
- A special case of the protocol is the ***Inter-ORB Protocol*** (***IIOP***), which is the GIOP applied to the TCP/IP transport layer.

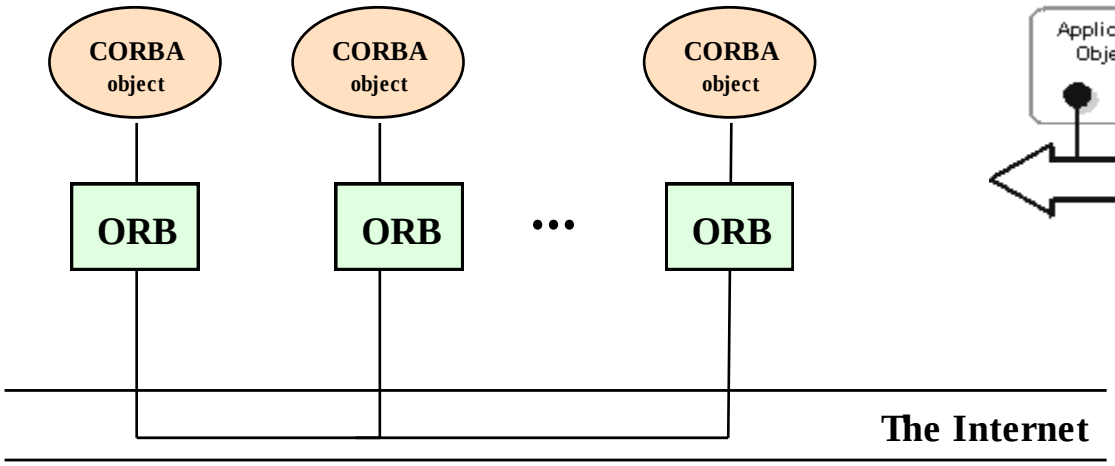
Inter-ORB Protocols

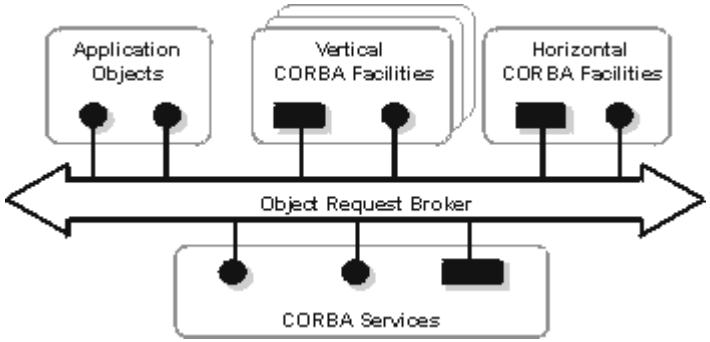
The IIOP specification includes the following elements:

1. **Transport management requirements**: specifies the connection and disconnection requirements, and the roles for the object client and object server in making and unmaking connections.
2. **Definition of common data representation**: a coding scheme for marshalling and unmarshalling data of each IDL data type.
3. **Message formats**: different types of message format are defined. The messages allow clients to send requests to object servers and receive replies. A client uses a Request message to invoke a method declared in a CORBA interface for an object and receives a reply message from the server.

Object Bus

An ORB which adheres to the specifications of the IIOP may interoperate with any other IIOP-compliant ORBs over the Internet. This gives rise to the term "***object bus***", where the Internet is seen as a bus that interconnects CORBA objects





ORB products

There are a large number of proprietary as well as experimental ORBs available:

(See CORBA Product Profiles, http://www.puder.org/corba/matrix/)

- Orbix IONA
- Borland Visibroker
- PrismTech's OpenFusion
- Web Logic Enterprise from BEA
- Ada Broker from ENST
- Free ORBs

Object Servers and Object Clients

- As in Java RMI, a CORBA distributed object is exported by an ***object server***, similar to the object server in RMI.
- An ***object client*** retrieves a reference to a distributed object from a naming or directory service, to be described, and invokes the methods of the distributed object.

CORBA Object References

- As in Java RMI, a CORBA distributed object is located using an ***object reference***.
- Since CORBA is language-independent, 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)

- For interoperability, OMG specifies a protocol for the abstract CORBA object reference object, known as the ***Interoperable Object Reference*** (***IOR***) protocol.
- An ORB compatible with the IOR protocol will allow an object reference to be
  - registered with and
  - retrieved from any IOR-compliant directory service.

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.

  The object key is used by an object server to locate the object.

Interoperable Object Reference (IOR)

The following is an example of the string representation of an IOR [5]:

```
IOR:000000000000000d49444c3a677269643a312e3000000
00000000001000000000000004c000100000000000015756c74
72612e6475626c696e2e696f6e612e696500000963000000002
83a5c756c7472612e6475626c696e2e696f6e612e69653a67
7269643a303a3a49523a6772696400 3a
```

The representation consists of the character prefix "`IOR:`" followed by a series of hexadecimal numeric characters, each character representing 4 bits of binary data in the IOR.

CORBA Naming Service

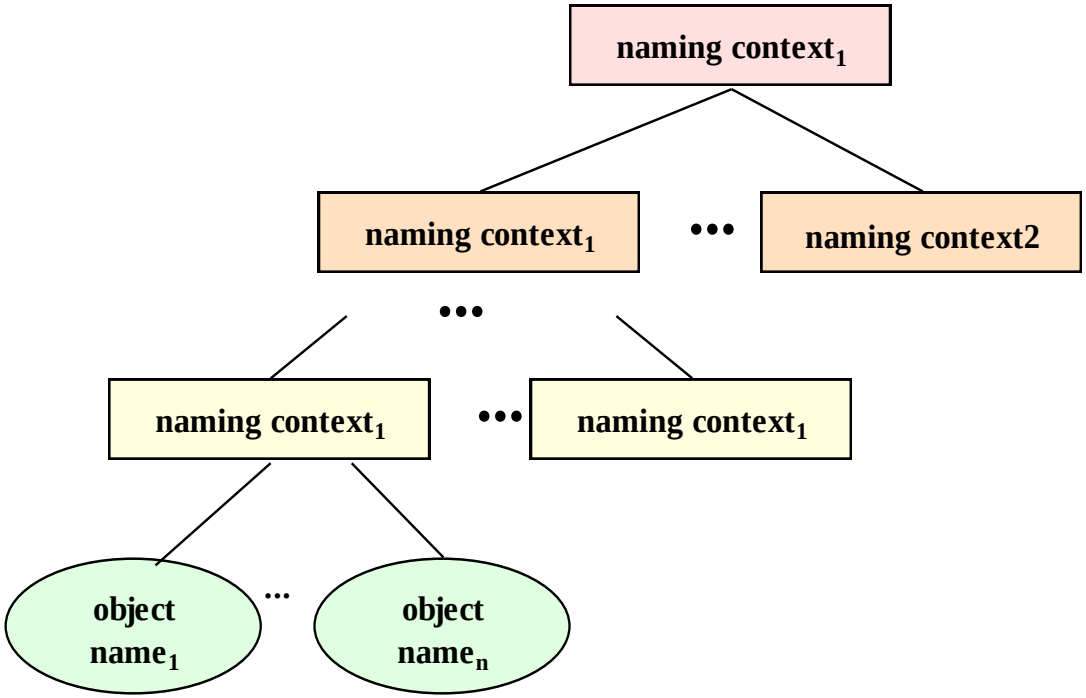
- CORBA specifies a generic directory service. The ***Naming Service*** serves as a directory for CORBA objects, and, as such, is platform independent and programming language independent.
- The Naming Service permits ORB-based clients to obtain references to objects they wish to use. It allows names to be associated with object references. Clients may query a naming service using a predetermined name to obtain the associated object reference.

CORBA Naming Service

- 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.
- To obtain a reference to the object, an object client requests the Naming Service to look up the object associated with the name. (This is known as ***resolving*** the object name.)
- The API for the Naming Service is specified in interfaces defined in IDL, and includes methods that allow servers to bind names to objects and clients to resolve those names.

CORBA Naming Service

To be as general as possible, the CORBA object naming scheme is necessarily complex. Since the name space is universal, a standard naming hierarchy is defined in a manner similar to the naming hierarchy in a file directory



A Naming Context

- A naming context correspond to a folder or directory in a file hierarchy, while object names corresponds to a file.
- The full name of an object, including all the associated naming contexts, is known as a *compound name*.
  - The first component of a compound name gives the name of a naming context, in which the second component is accessed. This process continues until the last component of the compound name has been reached.
- Naming contexts and name bindings are created using methods provided in the Naming Service interface.

A CORBA object name

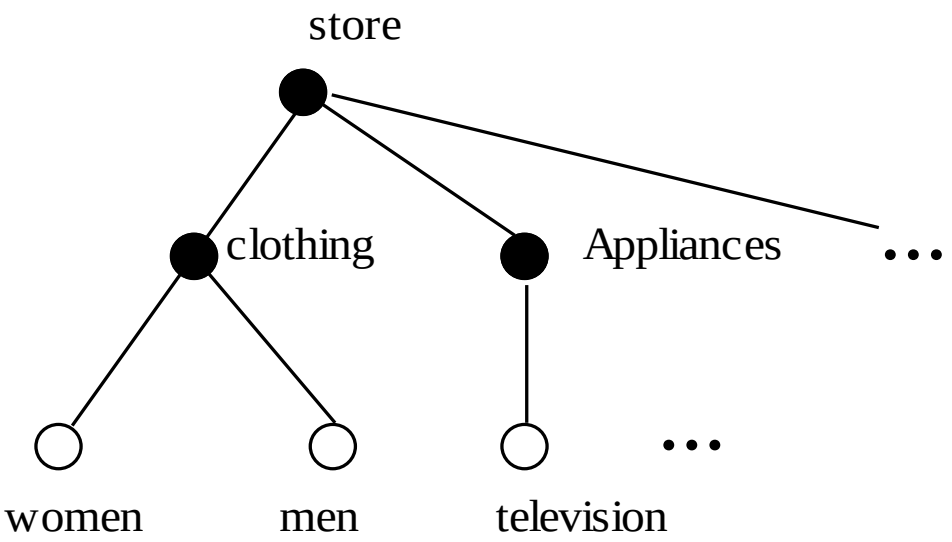
The syntax for an object name is as follows:

```
<naming context > …<naming context><object name>
```

where the sequence of naming contexts leads to the object name.

Example of a naming hierarchy

As shown, an object representing the men's clothing department is named store.clothing.men, where store and clothing are naming contexts, and men is an object name.



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 provides a URL to access a CORBA object.

CORBA Object Services

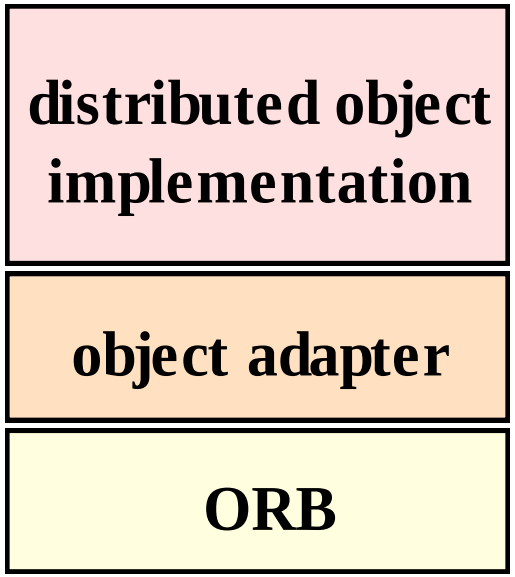
CORBA specifies services commonly needed in distributed applications, some of which are:

- ***Naming Service***:
- ***Concurrency Service***:
- ***Event Service***: for event synchronization;
- ***Logging Service***: for event logging;
- ***Scheduling Service***: for event scheduling;
- ***Security Service***: for security management;
- ***Trading Service***: for locating a service by the type (instead of by name);
- ***Time Service***: a service for time-related events;
- ***Notification Service***: for events notification;
- ***Object Transaction Service***: for transactional processing.

Each service is defined in a standard IDL that can be implemented by a developer of the service object, and whose methods can be invoked by a CORBA client.

Object Adapters

In the basic architecture of CORBA, the implementation of a distributed object interfaces with the skeleton to interact with the stub on the object client side. As the architecture evolved, a software component in addition to the skeleton was needed on the server side: an **object adapter**.



Object Adapter

- Object Adapter

http://www.cs.wustl.edu/~schmidt/PDF/POA.pdf

- An object adapter simplifies the responsibilities of an ORB by assisting 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 invoke 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, hence the word portable.

Java's CORBA Facility

JAVA IDL

Java IDL – Java's CORBA Facility

- IDL is part of the Java 2 Platform, Standard Edition (J2SE).
- The Java IDL facility includes a CORBA Object Request Broker (ORB), an IDL-to-Java compiler, and a subset of CORBA standard services.
- In addition to the Java IDL, Java 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

- package org.omg.CORBA – contains interfaces and classes which provides the mapping of the OMG CORBA APIs to the Java programming language
- package org.omg.CosNaming - contains interfaces and classes which provides the naming service for Java IDL
- org.omg.CORBA.ORB - contains interfaces and classes which provides APIs for the Object Request Broker.

Java IDL Tools

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

- idlj - the IDL-to-Java compiler (called idl2java in Java 1.2 and before)
- 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.
- tnameserv – an older Transient Java IDL Naming Service whose use is now discouraged.

Java RMI

CORBA (Common Object Request Broker Architecture): Java IDL

Distributed Component Object Model (DCOM)

Simple Object Access Protocol (SOAP) – web service

CASE STUDIES: APPLICATION DEVELOPMENT

Steps involved in developing CORBA applications

- Define an interface in IDL
- Map the IDL interface to any PL (e.g., use idlj for Java)
- For the server
  - Implement the interface `HelloImpl.java`
  - Develop the server `HelloServer.java`
- For the Client
  - Develop a client `HelloClient.java`

```
//Hello.idl
module HelloApp {
   interface Hello {
      string add(in string s);
      string say();
   };
};
```

```
> idlj -fall Hello.idl
HelloApp
   HelloOperations.java
   Hello.java
   HelloHelper.java
   HelloHolder.java

   HelloPOA.java
   _HelloStub.java
```

- Run the naming service, the server, and the client

Files Generated by idlj

`HelloOperations.java`

- Interface for method definition, shared by skeleton/stub

`Hello.java` → interface in 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.

`HelloHelper.java`

- 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.

Files Generated by idlj

`HelloHolder.java`

- Holds (contains) a reference to an object that implements the Hello interface.
- The class is used to handle an **out** or an **inout** parameter in IDL in Java syntax (In IDL, a parameter may be declared to be **out** if it is an output argument, and **inout** if the parameter contains an input value as well as carries an output value.)

`_HelloStub.java` → client stub

- the client-side proxy interfaces with the client object.
- It extends **org.omg.CORBA.portable.ObjectImpl** and implements the **Hello.java** interface.

`HelloPOA.java` → server skeleton

- Servant class HelloImpl.java extends corresponding this
- HelloPOA extends org.omg.PortableServer.Servant implements HelloApp.HelloOperations, org.omg.CORBA.portable.InvokeHandler

For the server

HelloImpl.java

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;
class HelloImpl extends HelloPOA {
  private ORB orb;
  private String message;
  public void setORB(ORB orb_val) {
    orb = orb_val;
  }
  public HelloImpl(String msg) {
        message = msg;
  }
  public String add( String s) {
    message = new String(message + s);
    return message;
  }
  public String say() {
    return message;
  }
}
```

```
//Hello.idl
module HelloApp {
  interface Hello {
    string sayHello();
    oneway void shutdown();
  };
};
```

```
> idlj -fall Hello.idl
HelloApp
   Hello.java
   HelloHolder.java
   HelloPOA.java
   HelloHelper.java
   HelloOperations.java
   _HelloStub.java
```

For the server

HelloServer.java

```
public class HelloServer {
  public static void main(String args[]) {
    try{

      ORB orb = ORB.init(args, null);       // create and initialize the ORB

      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate(); // get reference to rootpoa & activate the POAManager

      HelloImpl helloImpl = new HelloImpl("Hello, world!"); // create/register servant  with the ORB
      helloImpl.setORB(orb);

      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);             // get object reference from the servant
                                                        // get the root naming context
      org.omg.CORBA.Object objRef =orb.resolve_initial_references("NameService");
      // Use NamingContextExt which is part of the Interoperable, Naming Service (INS) specification.
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef); // bind the Object Reference in Naming

      String name = "Hello";
      NameComponent path[] = ncRef.to_name( name );
      ncRef.rebind(path, href);
      System.out.println("HelloServer ready and waiting ...");

      orb.run();                                // wait for invocations from clients
    }
      catch (Exception e) {
        System.err.println("ERROR: " + e);
        e.printStackTrace(System.out);
      }

      System.out.println("HelloServer Exiting ...");

  }
}
```

For the client

HelloClient.java

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, and invoking remote methods using the reference. The object's ***say*** method is invoked to receive a string, and the object's shutdown method is invoked to deactivate the service.

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class HelloClient
{
  static Hello hello;
  public static void main(String args[])
    {
      try{
         ORB orb = ORB.init(args, null);     // create and initialize the ORB
        org.omg.CORBA.Object objRef =        // get the root naming context
                          orb.resolve_initial_references("NameService");
        // Use NamingContextExt instead of NamingContext. This is
        // part of the Interoperable naming Service.
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        // resolve the Object Reference in Naming
        String name = "Hello";
        hello = HelloHelper.narrow(ncRef.resolve_str(name));

        System.out.println(hello.say( ));
        System.out.println(hello.add("Here is added information!!!"));
      }catch (Exception e) {
          System.err.println("ERROR : " + e) ;
          e.printStackTrace(System.out);
      }
  }
}
```

CORBA: Compile and Run the Server

Map into Java by running the IDL-to-java:

```
idlj -fall Hello.idl
```

Compile the server:

```
javac HelloServer.java HelloImpl.java HelloApp/*.java
```

Start the Java Object Request Broker Daemon (orbd) on a machine X

```
orbd -ORBInitialPort 20000 -ORBInitialHost X &
```

Start HelloServer on another machine Y

```
java HelloServer -ORBInitialPort 20000 -ORBInitalHost X
```

CORBA: Compile and Run the Client

Map into Java by running the IDL-to-java:

```
idlj -fall Hello.idl
```

Compile the client :

```
javac HelloClient.java HelloApp/*.java
```

Start HelloClient on still another machine Z

```
java HelloClient -ORBInitialPort 20000 -ORBInitalHost X
```

Hello, world!

Hello, world!Here is added information!!!

> java HelloClient …

Hello, world!Here is added information!!!

Hello, world!Here is added information!!!Here is added information!!!

> java HelloClient …

Hello, world!Here is added information!!!Here is added information!!!

Hello, world!Here is added information!!!Here is added information!!!Here is added information!!!

> ps
> kill -9 1611 #orbd-pid
> java HelloClient …

Feb 21, 2013 2:33:45 PM
com.sun.corba.se.impl.transport.SocketOrChannelConnectionImpl <init>
WARNING: "IOP00410201: (COMM_FAILURE) Connection failure: socketType: IIOP_CLEAR_TEXT; hostname: 127.0.0.1; port: 20000"
…

Summary-1

Common Object Request Broker Architecture (CORBA)

- The key topics introduced with CORBA are:
  - The basic CORBA architecture and its emphasis on object interoperability and platform independence
  - Object Request Broker (ORB) and its functionalities
  - The Inter-ORB Protocol (IIOP) and its significance
  - CORBA object reference and the Interoperable Object Reference (IOR) protocol
  - **CORBA Naming Service** and the **Interoperable Naming Service** (**INS**)
  - Standard CORBA **object services** and how they are provided.
  - **Object adapters**, **portable object Adapters** (**POA**) and their significance.

Summary-2

a specific CORBA facility based on Java IDL

- The key topics introduced with **Java IDL** are:
  - It is part of the Java™ 2 Platform, Standard Edition (J2SE)
  - Java packages are provided which contain interfaces and classes for CORBA support
  - Tools provided for developing a CORBA application include ***idlj*** (the IDL compiler) and ***orbd*** (the ORB and name server)
  - An example application Hello
  - Steps for compiling and running an application.
  - Client callback is achievable.
- CORBA tookits and Java RMI are comparable and alternative technologies that provide distributed objects. An application may be implemented using either technology. However, there are tradeoffs between the two.