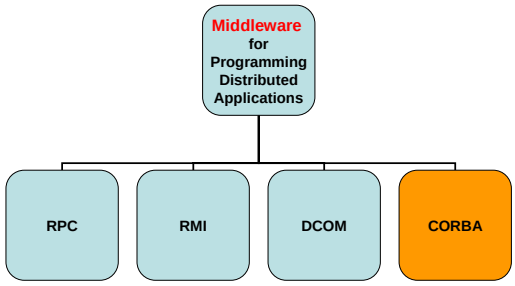
Desarrollo de Aplicaciones en Red

José Rafael Rojano Cáceres
http://www.uv.mx/rrojano

Session plan

- ***General vision***
- ***Middleware***
- ***OMA***
- ***Corba***
  - **IDL**
  - **ORB**
  - **IIOP**
- ***Examples***

What's Corba?



RPC vs. Corba

- ***Corba share the same model that RPC***
- ***Instead of RPC, Corba defines an complete architecture and identify with detail the elements.***
- ***RPC is a mechanism of communication***
- ***Corba is an reference architecture that includes an mechanism of communication***

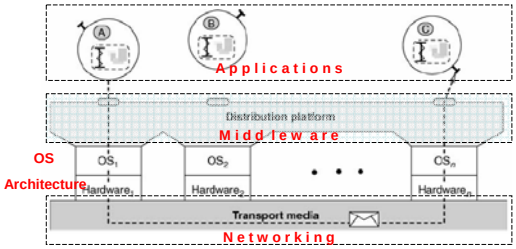
Middleware

- ***In order to provide a solution to all the problems generated by Distributed Systems we got middleware.***
- ***Middleware offers general services that support distributed execution of applications.***

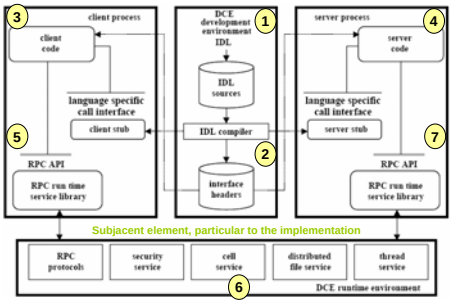
Middleware task

- Object model support: Middleware should offer mechanisms to support the concepts incorporated in the object model.
- Operational interaction: Middleware should allow the operational interaction between two objects. The model used is the method invocation for an object in oriented programming language.
- Remote interaction: Middleware should allow the interaction between two objects located in different address spaces.
- Distribution transparency: From the standpoint of the program, interaction between objects is identical for both local and remote interactions.
- Technological independence: The middleware supports the integration of different technologies.

The structure of a middleware platform



DCE Middleware



Summary

- ***Middleware often is saw for the programmer as a API (Application Programming Interface).***
- ***Middleware takes care of differences in the architecture like byte ordering.***
- ***Middleware offers communication with different languages. This through an IDL language.***
- ***Middleware is a set of models that offer a programming environment simple, consistent, and integrated to keep simple the process of design, programming, and management of applications.***

OMA

Object Management Architecture

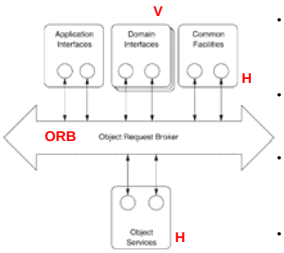
OMA (1)

- **The Object Management Architecture (OMA) is a standard, which describes a general platform for the development of distributed, object-oriented applications.**
- **The Common Object Request Broker Architecture (CORBA) is a standard also, which is an specialization of OMA and describes an actual middleware platform**

OMA (2)

- ***OMA defines:***
  - **An object model, which makes differentiation between**
    - Object semantics
      - Characteristic visible to client
    - Object implementation
      - How to execute objects
  - **A reference architecture defines:**
    - Relationship between objects

OMA Reference Architecture



- **Object services:** This category combines the horizontal system services that are application-independent and can be used in different contexts. Examples of object services include naming, trading, and security services.
- **Common facilities:** The common facilities provide horizontal end user services that are typically required in different application contexts. An example is the printing service.
- **Domain interfaces:** The domain interfaces represent vertical services for special application areas. Examples of domain interfaces are medical, telecommunications, and financial services.
- **Application interfaces:** The application interfaces represent application-specific services. In contrast to the three other categories, application interfaces are not included in the OMG standardization efforts.

© Morgan Kaufmann, Distributed Systems a middleware approach

13

CORBA™

Common Object Request Broker Architecture

14

What's CORBA (1)

- The Common Object Request Broker Architecture (CORBA) was first published in 1990 by the Object Management Group (OMG), a non-profit organization that was founded in 1989 to integrate distributed applications based on a variety of existing technologies.
- CORBA standardizes interfaces and semantics for object-oriented middleware.
- It includes a specification for the Object Request Broker (ORB), a software library with standardized CORBA object interfaces that allows clients and targets to communicate with each other across a network in a well-defined way. In addition, CORBA automatically applies a range of useful services to communications. After the ORB is initialized, all CORBA objects can be invoked by applications like local software objects.

http://www.mico.org/

15

What's CORBA (2)

- Corba is a set of specifications (not implementations) that let object to communicate between different languages and different platforms.
- The CORBA [Corba] cation v1.1 was introduced in 1[Object Model] [Reference Model] give a standard mechanism for objects to communicate across[ork
- Later releases spec were CORBA 2.0 in 1994 and CORBA 3.0 in

First global vision of Corba

16

Terminology (1)

- ***A Object Model aims to provide a systematic representation of the problem area based on: abstraction, modularity, encapsulation, and hierarchy.***
- ***Object:*** An object is therefore a self-contained problem area.
  - *The objects have certain elements showed in the next slide.*

17

Terminology (2)

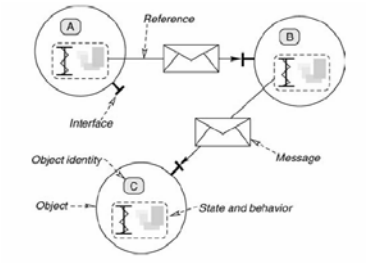
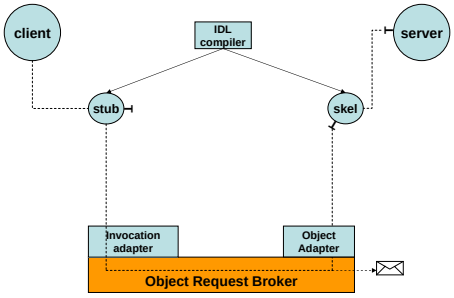


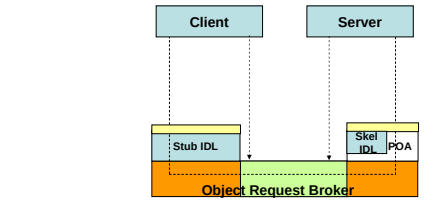
FIGURE 2.5 Decomposition of a problem domain into a set of objects.

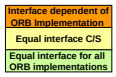
18

Corba components (1)



Corba components (2)





Provider spec (private)

OMG spec (public)

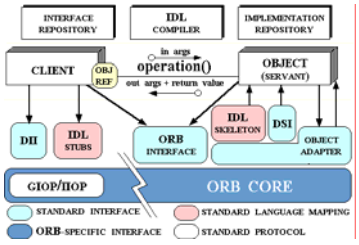
ORB

Object Request Broquer

ORB

- The Object Request Broker is the core element in the Corba architecture.
- The ORB provides a mechanism for transparently communicating client requests to target object implementations.
- The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.
- In the next slide you can see an conceptual overview of the ORB

Corba ORB Architecture



http://www.cs.wustl.edu/~schmidt/corba-overview.html

Components ORB (1)

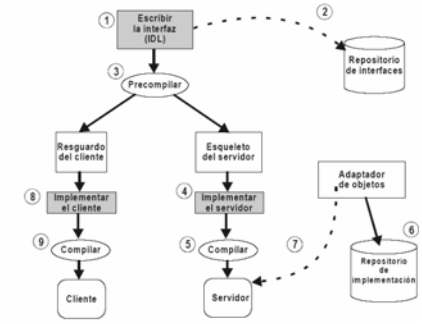
- An Object is a CORBA programming entity that consists of an identity, an interface, and an implementation, which is known as a Servant.
  - *Servant, This is an implementation programming language entity that defines the operations that support a CORBA IDL interface. Servants can be written in a variety of languages, including C, C++, Java, Smalltalk, and Ada.*
- ORB Interface, is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface described below.

Components ORB (2)

- **Dynamic Invocation Interface (DII), This interface allows a client to directly access the underlying request mechanisms provided by an ORB. Applications use the DII to dynamically issue requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs (which only allow RPC-style requests), the DII also allows clients to make non-blocking deferred synchronous (separate send and receive operations) and oneway (send-only) calls.**
- **Dynamic Skeleton Interface (DSI), This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.**
- **Object Adapter, This assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects).**

25

Static invocation



26

IDL

Interface Definition Language

27

IDL (1)

- The *Interface Definition Language* (IDL) is used to specify object interfaces independently of a specific programming language.
- IDL is the basis for the separation of the interface and the implementation of an object.
- **OMG**-IDL is a *declarative language* it contains no algorithmic constructs for the description of loops, branching, and so forth.
- Its syntax is extensively based on that of C++, but it includes some additional constructs to accommodate the special characteristics of distributed environments (for example, the identification of parameters as input or output parameters).

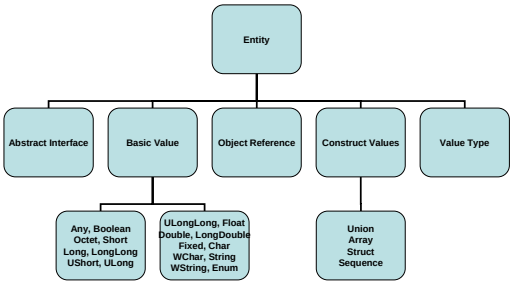
28

IDL (2)

- ***Don't forget there are different IDL not only for Corba.***
- ***Currently OMG defines IDL for: C, C++, Smalltalk, Phyton, Ada 95, Cobol, PL/1 and Java.***
- ***Each IDL files is mapped to the correspondent stub and skeleton.***

29

Entities IDL



30

IDL syntax

```
module <identification> {
    <type declaration>
    <constant declaration>
    <exception declaration>

     Interface <identification>[:<inheritance>]{
         <type declaration>
         <constant declaration>
         <attributes declaration>
         <exception declaration>
         [<return type >] <identification>(<list of arguments>) [raises <exception>]
         [<return type >] <identification>(<list of arguments>) [raises <exception>]
         .....
     };

     Interface <identification>[:<inheritance>]
    { ..... }
    ....
};
```

http://www.omg.org/technology/documents/idl2x_spec_catalog.htm

IIOP

Internet Inter ORB Protocol

IIOP

- ***This is the protocol used to transmit the message through ORB.***
- ***This protocol is based in GIOP (General Inter-ORG Protocol) which is an specification***
- ***IIOP defines:***
  - **Requirements for transport layer**
  - **The CDR (Common Data Representation)**
  - **The Message format**
- ***IIOP ←→ JRMP***

Example

© Hello is SUN's property

IDL file

```
1  module HelloApp
   {
2    interface Hello
     {
3      string sayHello();
       oneway void shutdown();
     };
   };
```


Compiling IDL files

- ***Idlj –fall Hello.IDL***
  - **HelloPOA.java** This abstract class is the stream-based server skeleton, providing basic CORBA functionality for the server. It extends org.omg.PortableServer.Servant, and implements the InvokeHandler interface and the HelloOperations interface. The server class, HelloServant, extends HelloPOA.
  - **_HelloStub.java** This class is the client stub, providing CORBA functionality for the client. It extends org.omg.CORBA.portable.ObjectImpl and implements the Hello.java interface.
  - **Hello.java** This interface contains the Java version of our IDL interface. The Hello.java interface extends org.omg.CORBA.Object, providing standard CORBA object functionality. It also extends the HelloOperations interface and org.omg.CORBA.portable.IDLEntity.
  - **HelloHelper.java** This class provides auxiliary functionality, notably the narrow() method required to cast CORBA object references to their proper types. The Helper class is responsible for reading and writing the data type to CORBA streams, and inserting and extracting the data type from Anys. The Holder class delegates to the methods in the Helper class for reading and writing.
  - **HelloHolder.java** This final class holds a public instance member of type Hello. Whenever the IDL type is an out or an inout parameter, the Holder class is used. It provides operations for org.omg.CORBA.portable.OutputStream and org.omg.CORBA.portable.InputStream arguments, which CORBA allows, but which do not map easily to Java's semantics. The Holder class delegates to the methods in the Helper class for reading and writing. It implements org.omg.CORBA.portable.Streamable.
  - **HelloOperations.java** This interface contains the methods sayHello() and shutdown(). The IDL-to-Java mapping puts all of the operations defined on the IDL interface into this file, which is shared by both the stubs and skeletons.

Servant

```
// HelloServer.java
// Copyright and License
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;
  public void setORB(ORB orb_val) {
    orb = orb_val;
  }

  // implement sayHello() method
  public String sayHello() {
    return "\nHello world !!\n";
  }

  // implement shutdown() method
  public void shutdown() {
    orb.shutdown(false);
  }
}
```

(1)

(2)

Establece los valores para el ORB

(3)

Server

```
public class HelloServer {
  public static void main(String args[]) {
    try{
      // create and initialize the ORB
      ORB orb = ORB.init(args, null);
      // get reference to rootpoa & activate the POAManager
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();
      // create servant and register it with the ORB
      HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);
      // get object reference from the servant
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);
      // 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 ...");
      // wait for invocations from clients
      orb.run();
    }
      catch (Exception e) {
        System.err.println("ERROR: " + e);
        e.printStackTrace(System.out);
      }
      System.out.println("HelloServer Exiting ...");

  }
}
```


Client

```
// Copyright and License

import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class HelloClient
{
  static Hello helloImpl;
  public static void main(String args[])
    {
      try{
        // create and initialize the ORB
         ORB orb = ORB.init(args, null);
        // get the root naming context
        org.omg.CORBA.Object objRef =
            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";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
        System.out.println("Obtained a handle on server object: " + helloImpl);
        System.out.println(helloImpl.sayHello());
        helloImpl.shutdown();
         } catch (Exception e) {
          System.out.println("ERROR : " + e) ;
          e.printStackTrace(System.out);
          }
    }
}
```


Running the example

- *idlj –fall Hello.idl*
- *javac HelloApp/*.java*
- *javac HelloServer.java*
- *javac HelloClient.java*
- *start orbd -ORBInitialPort 1050 -ORBInitialHost localhost*
- *start java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost*
- *java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost*

Reference

- *Puder, Römer, & Pilhofer - Distributed Systems a middleware approach, 2006*
- *Abian, Miguel. Java y las redes, 2004.*
- *Horstman Cay, Java 2, 2003.*
- *Sun, J2SE documentation*
- *Sun, JSEE documentation*
- *http://www.omg.org/technology/documents/corba_spec_catalog.htm*
- *http://docs.sun.com/source/817-5445/agcorba.html*
- *http://www.omg.org/docs/formal/04-03-01.pdf*