Diseño Y Aplicaciones de Sistemas Distribuidos

La arquitectura CORBA

Joan Vila

DISCA / UPV

Departament d'Informàtica de Sistemes i Computadors

Universitat Politècnica de València

DISCA
DEPARTAMENT D'INFORMÀTICA DE SISTEMES I COMPUTADORS

La arquitectura CORBA

- **Índice**
  - La arquitectura CORBA
  - Desarrollo de aplicaciones CORBA
    - Definición y procesamiento de la interfaz IDL
    - Factorías de objetos
    - El problema de la herencia múltiple
  - El servicio de nombres

La arquitectura CORBA

- **CORBA: *Common Object Request Broker Architecture***
  - Proyecto de *middleware* que define una arquitectura estándar basada en el **modelo de objetos**, que permite interoperabilidad en aplicaciones distribuidas en un entorno completamente ***heterogéneo.***
  - Definido por el consorcio de fabricantes *OMG* (Object Management Group) compuesto por las siguientes compañías: *SunSoft, IONA, DEC, HP, NCR, HyperDesk, ObjectDesign*

    http://www.omg.org

    http://www.omg.org/docs/formal/02-12-02
  - **Enfoque** de OMG segun el libro "Object Management Architecture Guide“:
    - *"To adopt interface and protocol specifications that define an **object management architecture supporting interoperable applications based on distributed interoperable objects**. The specifications are to be based on existing technology that can be demonstrated to satisfy OMG's Technical Objectives"*

La arquitectura CORBA

- **Componetes de CORBA**

  OMG desarrolla:

  – Un modelo conceptual de un bus software al que pueden "conectarse" objetos en un sistema distribuido.

  – Una arquitectura de referencia, denominada *Object Management Architecture* (OMA) con los siguientes componentes:

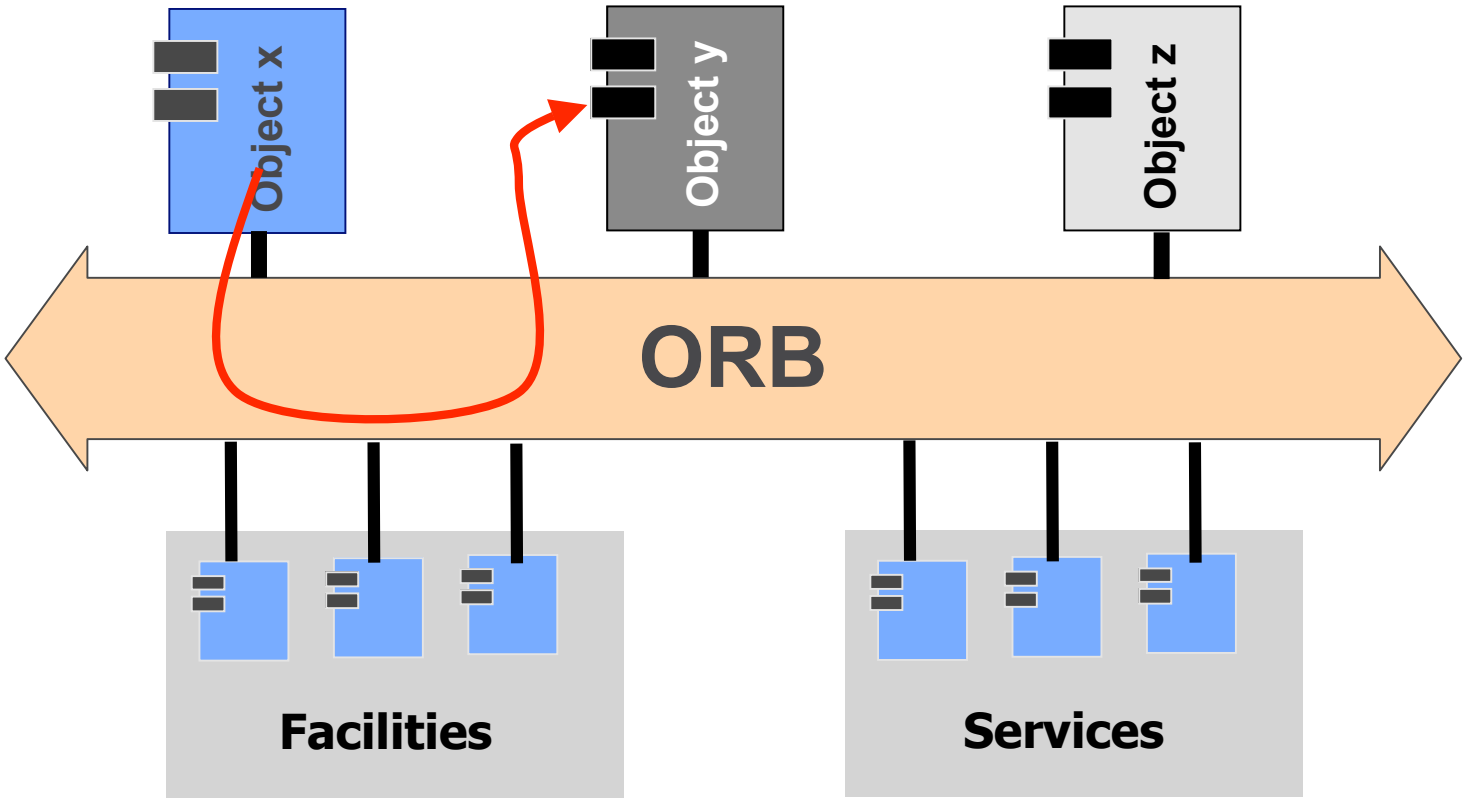
  - **Object Request Broker (ORB):** núcleo de OMA. Bus de comunicación de objetos.
  - **Application Objects (AO):** los objetos que se conectan al ORB
  - **Object Services (OS):** colecciones de servicios de nivel de sistema que aumentan la funcionalidad de ORB.

    Ejemplos: *Servicio de nombres, Trading Service, Servicio de tiempo, Servicio de eventos*
  - **Common Facilities (CF):** servicios para entornos de aplicación muy específicos.

    Ejemplos: *comercio-e, gestión de bases de datos, agentes móviles, negocios, ...*

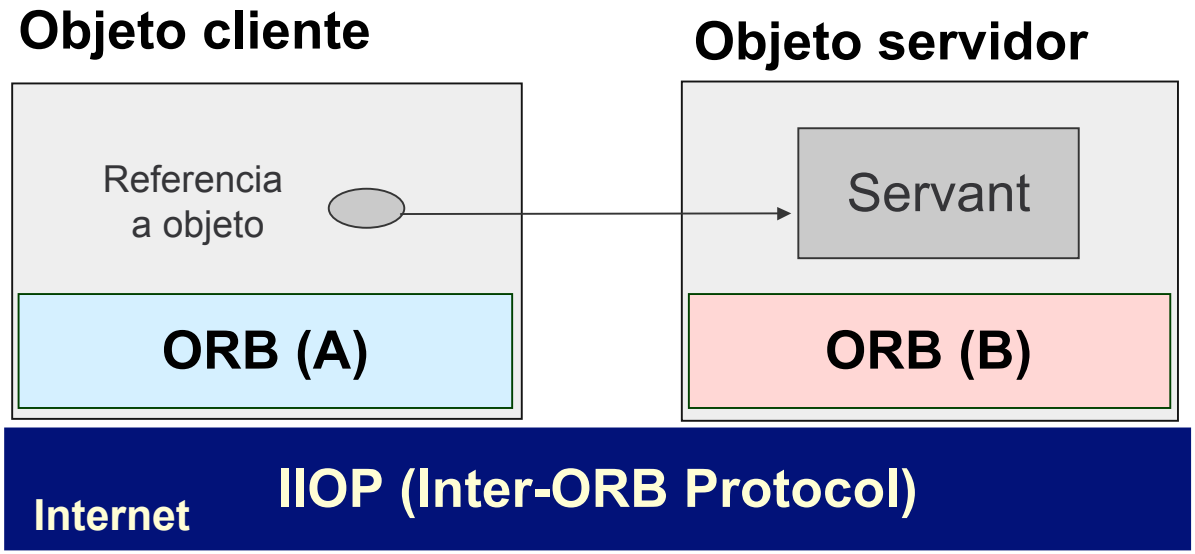
La arquitectura CORBA

- La *Object Management Architecture* (OMA)



La arquitectura CORBA

- **CORBA 2.0: la arquitectura Inter-ORB**
  - **CORBA 1.1** no especificaba la implementación de ORB y, por tanto, resultaba difícil la comunicación entre dos ORBs de distinto fabricante. El resultado fue un cierto grado de portabilidad, pero no de interoperabilidad.
  - **CORBA 2.0** establece **IIOP** como protocolo estandarizado para inter-comunicación entre ORBs. Todo ORB conforme a CORBA debe implementarlo o proporcionar un semi-puente a el.



La arquitectura CORBA

- **CORBA 2.0: la arquitectura Inter-ORB**

  Los componentes básicos de esta arquitectura son:

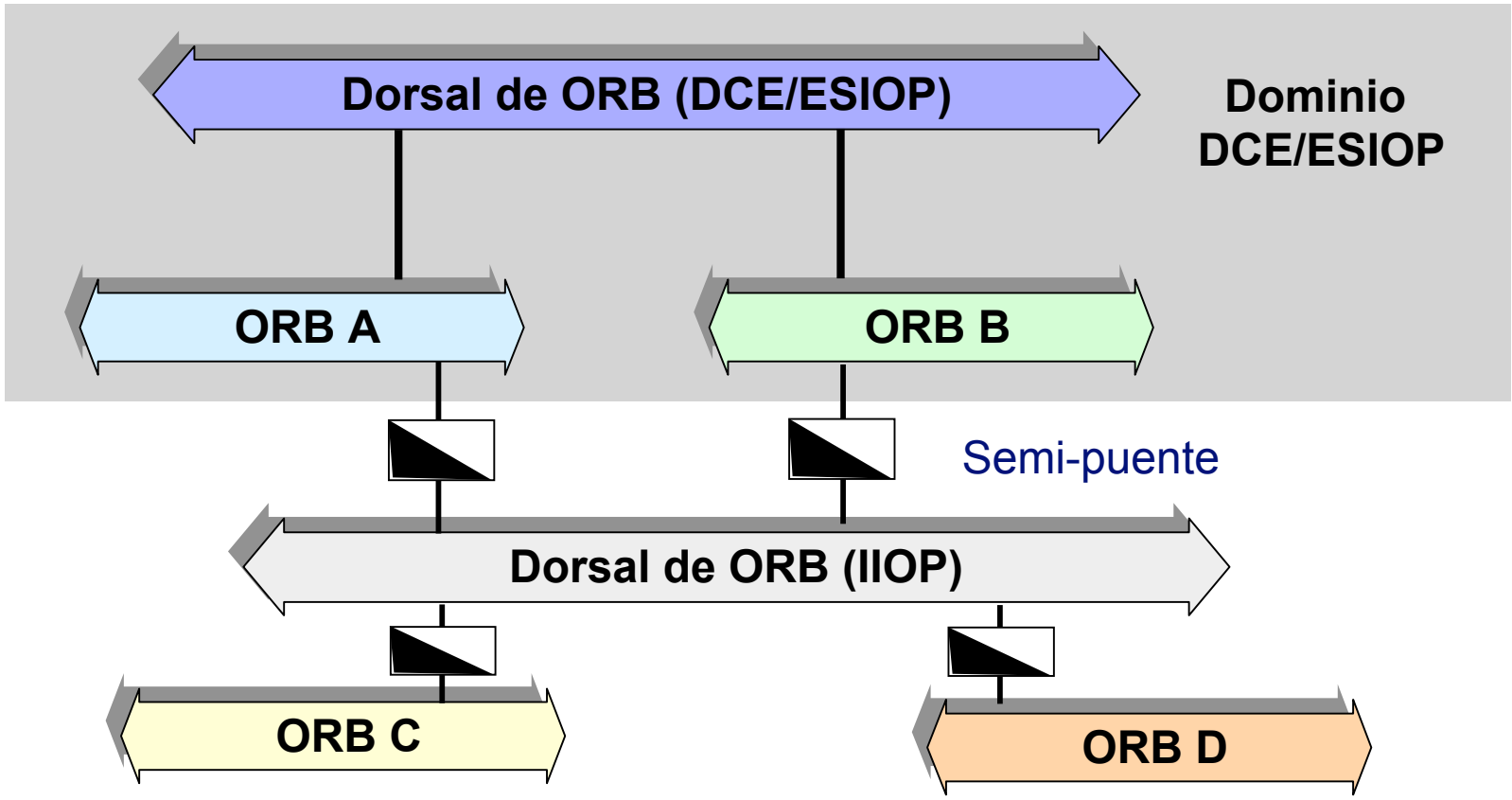
  – **GIOP (*General Inter-ORB Protocol*)**: especifica:
    - Un conjunto de mensajes
    - **CDR**: Representaciones de Datos Comunes
    - **IOR**: Referencias a Objetos Inter-Operables

    para comunicación entre ORBs

  – **IIOP (*Internet Inter-ORB protocol*):** especifica como intercambiar mensajes GIOP en una red TCP/IP. Esto permite utilizar Internet como una dorsal de ORB.
    - **ESIOP (*Evironment-Specific Inter-ORB Protocol*):** Protocolos de comunicación Inter-ORB sobre redes específicas como por ejemplo DCE (DCE/ESIOP).

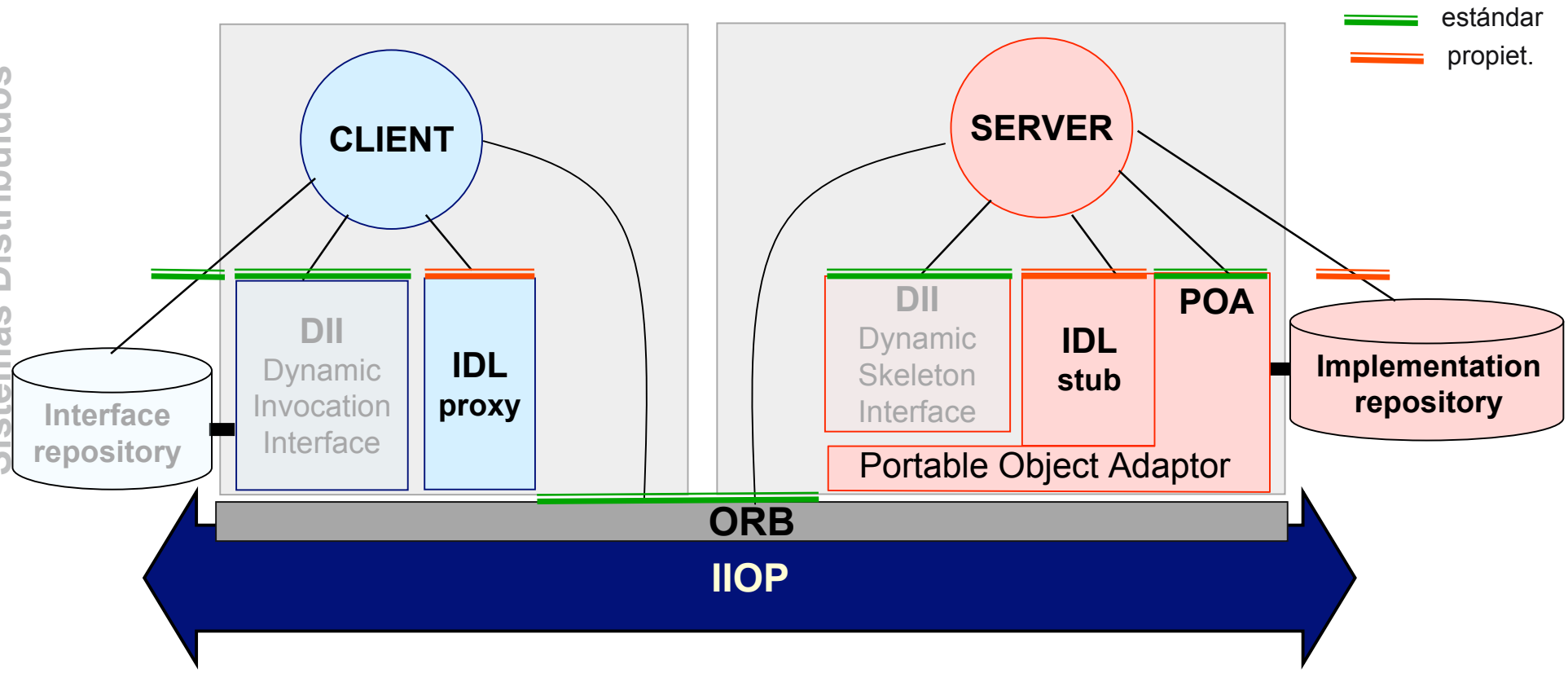
La arquitectura CORBA

- **CORBA 2.0: la arquitectura Inter-ORB**



La arquitectura CORBA

- **Interfaces CORBA**
  - La especificación CORBA define distintas interfaces para que clientes y servidores realicen la comunicación entre objetos.



La arquitectura CORBA

- **Interfaces estáticas / interfaces dinámicas**
  - El mecanismo estático asume que los clientes conocen la interfaz de los servicios en tiempo de compilación y que esta interfaz no cambia. Aunque esto es suficiente para la mayoría de las aplicaciones, restringe la utilización de nuevos servicios que puedan aparecer durante el ciclo de vida de una aplicación o que una interfaz pueda cambiar.
- **Interfaces estáticas: proxy / stub IDL**
  - Se generan automáticamente a partir de una **interfaz IDL.**
  - Presentan una interfaz derivada de la definición IDL del servicio y definen cómo se utiliza un servicio
  - La interfaz se denomina también “estática”, puesto que se define en tiempo de compilación y ya no cambian en tiempo de ejecución.
  - El **proxy IDL** se monta junto con el programa cliente.
  - El **stub IDL** se monta junto con el programa servidor.

La arquitectura CORBA

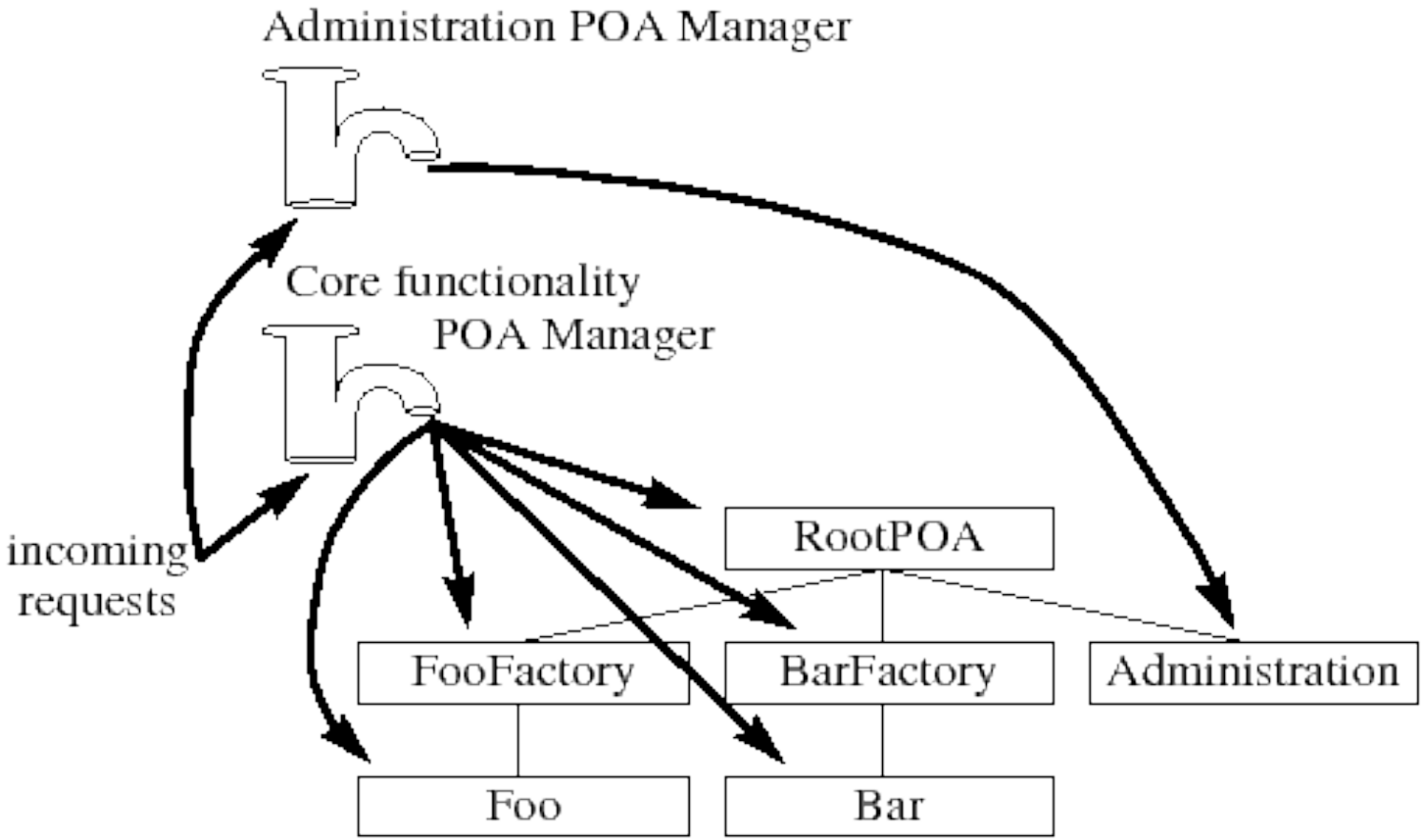
- **La interfaz POA: Portable Object Adapter**
  - El POA es la conexión de un objeto al bus ORB.
  - Un POA gestiona un conjunto de objetos.
  - El programa servidor registra en el POA todas las instancias de objetos listos para ser usados. Una vez ocurre esto, el POA gestiona las peticiones que llegan al servidor. Sus funciones incluyen:
    - Gestionar los identificadores de objetos
    - Recibir y gestionar peticiones
    - Establecer el modelo de concurrencia
    - Implementar las políticas de objetos persistentes y transitorios
- **Objetos transistorios y persistentes**
  - Un POA tiene asignado un valor de política TRANSIENT o PERSISTENT.
    - **TRANSIENT**: no mantiene el estado entre activaciones.
    - **PERMANENT**: no mantiene el estado entre activaciones. Las referencias a objetos persistentes continúan siendo válidas más allá de la vida del POA.

La arquitectura CORBA

- **El POA: Activación/Desactivación de Servants**
  - Para que un objeto pueda recibir invocaciones de los clientes debe ser activado.
    - Esto informa al ORB de que *servant* en particular encarna a un objeto.
  - La activación le asigna un id a un *servant*.
    - Este id está también incrustado en las referencias al objeto.
  - La activación puede hacerse de dos modos:
    - **Implícita**: creando una instancia e invocando ***_this***. Se usa para objetos TRANSITORIOS.
    - **Explícita**: ***poa.activate_object***. Se usa para objetos PERSISTENTES.

La arquitectura CORBA

- **El POA manager**
  - Un grupo de objetos puede tener su gestor o POA manager.



La arquitectura CORBA

- **Interfaces dinámicas**
  - Se generan a partir del **Interface Repository**
  - **Dynamic Invocation Interface (DII)** – Permite especificar y construir invocaciones en tiempo de ejecución en vez de invocarlas mediante "montado" en tiempo de compilación a través del proxy. Las operaciones de DII incluyen: **create_ request, invoke, send, get_response.**
  - **Dynamic Skeleton Interface (DSI)** – Parte del servidor análoga a la parte del cliente. El DSI inspecciona las invocaciones que recibe para determinar el objeto y método a invocar. Permite a un servicio asumir el rol de otro servicio.
  - Un objeto no puede distinguir las invocaciones que le llegan por via estática o dinámica.
- **El Interface Repository**
  - El Interface Repository permite a un servicio registrar su interfaz en un catálogo.
  - Permite a los clientes buscar interfaces en tiempo de ejecución y localizar servicios desconocidos en tiempo de compilación.
  - La interfaz **proporciona funciones para listar y buscar nuevos servicios que después serán invocados de forma dinámica (DII).**

La arquitectura CORBA

- **El Implementation Repository**
  - Proporciona **soporte para binding indirecto de referencias a objetos persistentes**.
  - Flexibiliza el acoplamiento entre un cliente y un servidor, permitiendo al servidor cambiar de ubicación sin afectar al cliente.
  - Proporciona la habilidad de **arrancar servidores por demanda**.
  - Referencias Indirectas
    - Cuando un servidor utiliza el IMR, las referencias a objetos devueltas por el POA se refieren al IMR en vez de al servidor.
    - Cuando un cliente utiliza esta referencia, el IMR recibe la petición, activa el servidor (si es necesario) y devuelve una nueva referencia al cliente que identifica al servidor (incluido su host y port).
    - El cliente establece una conexión con el servidor utilizando la nueva referencia y comunicando directamente con el servidor (sin mediar el IMR).
    - Si el servidor falla, el cliente debe contactar con el IMR otra vez, que debe rearrancar el servidor y permitir al cliente reanudar sus actividades.
  - Interfaz propietaria
    - La especificación CORBA no estandariza como interactúan los servidores y el *Implementation Repository*, sólo sugiere la funcionalidad que los vendedores deben implementar.

La arquitectura CORBA

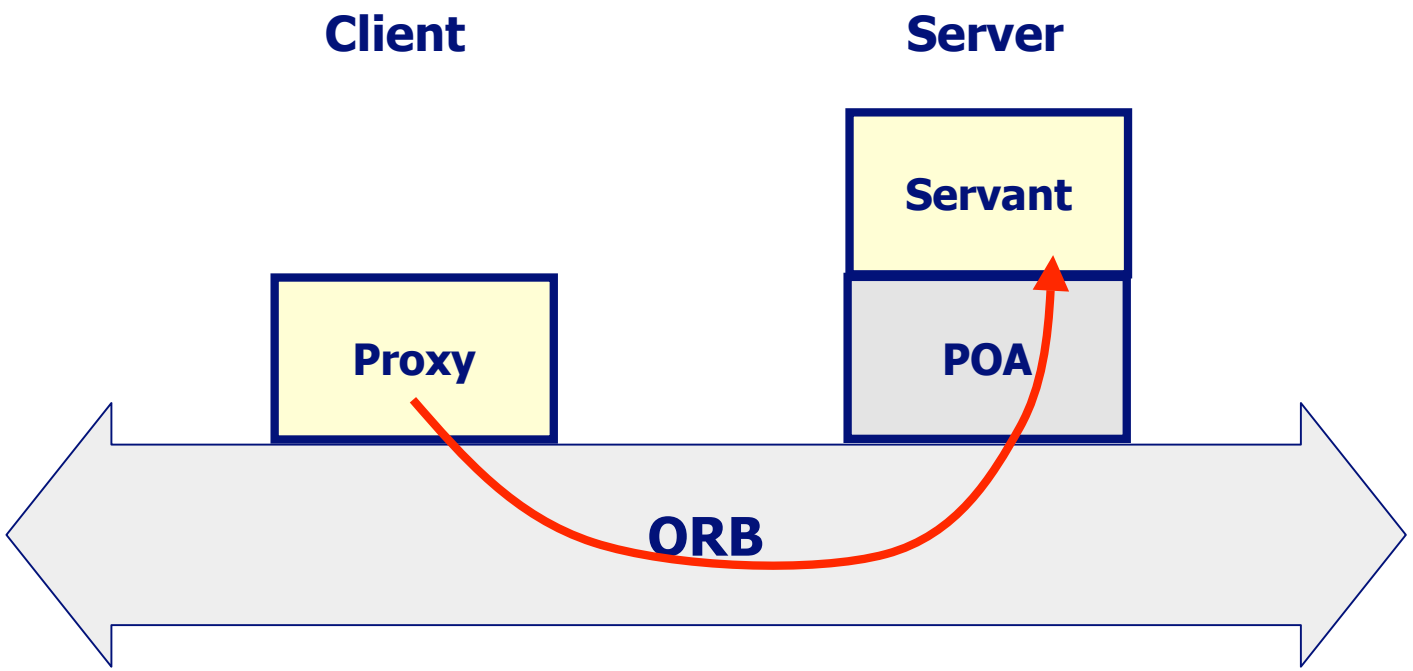
- **La interfaz ORB:Object Request Broker**
  - Tiene una interfaz directa mínimaq sobre la aplicación. Incluye funciones para:
    - Derivar IOR (Interoperable Object References) de strings y viceversa
    - Registrar y obtener referencias a servicios iniciales (servicio de nombres) y arrancarla

La arquitectura CORBA

- **Índice**
  - La arquitectura CORBA
  - Desarrollo de aplicaciones CORBA
    - Definición y procesamiento de la interfaz IDL
    - Factorías de objetos
    - El problema de la herencia múltiple
  - El servicio de nombres

Desarrollo de aplicaciones CORBA

- **Objetos en CORBA**
  - Objeto CORBA: Objeto con interfaz IDL. Tiene dos representaciones
    - **Proxy** en el cliente
    - **Servant**: en el servidor. Es la parte que implementa el código de aplicación de un servicio. Está separado del "código CORBA".



Desarrollo de aplicaciones CORBA

Definición de interfaces

CORBA estandariza IDL como forma de definir interfaces.

- Ver transparencias IDL

– Una especificación IDL es un contrato entre cliente y servidor:
  - IDL especifica la sintaxis de las interfaces
  - La semántica se puede expresar como comentarios

– IDL es independiente de lenguajes de programación y sistemas operativos
  - La correspondencia de IDL a Java o C++ la realiza el compilador

– Una especificación en IDL comprende:
  - Declaraciones de módulos
  - Declaraciones de interfaces
    – Soporta herencia múltiple
    – Declaraciones de operaciones y atributos
  - Declaraciones de tipos de datos, constantes y excepciones

– También soporta las características de preproceso de C++IDL

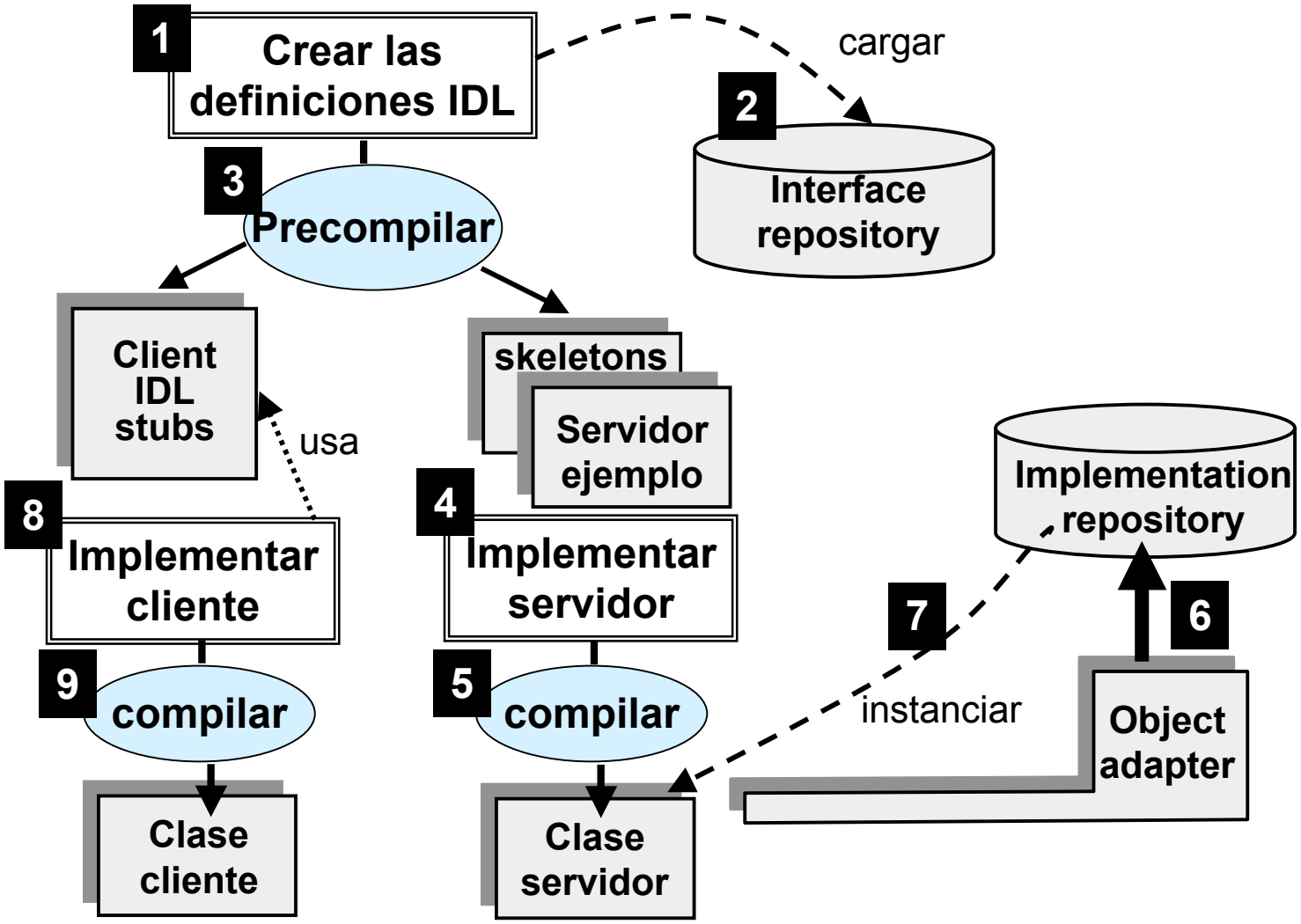
Desarrollo de aplicaciones CORBA

- **IDL (*Interface Definition Language*)**

Un ejemplo de interfaz:

```
//IDL: definicion de una matriz
interface matriz {
//atributos
readonly attribute short  filas;
readonly attribute short  columnas;
//excepciones
exception overflow{};
//operaciones
void poner(in short fila, in short columna, in long valor)
        raises (overflow);
long obtener(in short fila, in short columna);
        raises (overflow);
};
```

Desarrollo de aplicaciones CORBA



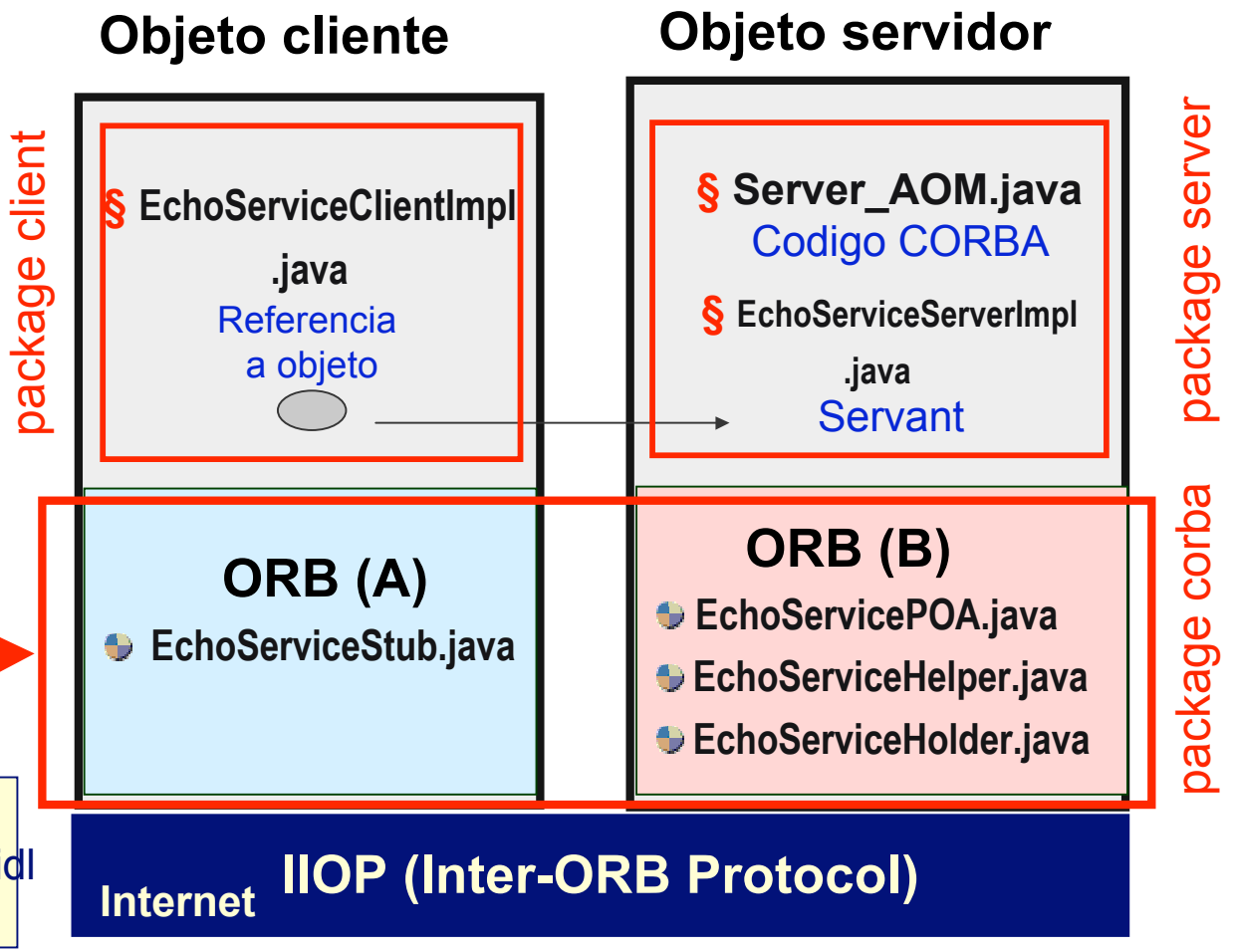
Desarrollo de aplicaciones CORBA

Echo.idl

```
module corba {
   interface EchoService {
      string echo(in string x);
   };
};
```

jidl

§: Ficheros a realizar
Ficheros generados por jidl



Desarrollo de aplicaciones CORBA

- **El procesador jidl**

  Produce los siguientes ficheros:

  - EchoService**.java** file://Echo.java.pdf

    Versión Java de la interfaz IDL. Está vacío, pero es subclase de:

    - **org.omg.CORBA.Object**: proporciona funcionalidad de objeto CORBA estándar.
    - **EchoOperations:** definido en el siguiente fichero
    - Este interfaz es el que implementa el stub del cliente

  - EchoService**Operations.java**

    Es el verdadero interfaz java.

  Se adopta esta estructura para facilitar la delegación.

Desarrollo de aplicaciones CORBA

- **El procesador jidl (ii)**

  y también los siguientes ficheros (que no hay que modificar):

  – EchoService**POA.java**

    En Orbacus integra el stub del servidor

  – EchoService**Helper.java**

    Clase final que proporciona funcionalidad extra, fundamentalmente el método *narrow* para convertir referencias CORBA a su correspondientes tipos.

  – EchoService**Holder.java**

    Proporciona operaciones para los argumentos out e inout de CORBA que no se ajustan fácilmente a la semántica de Java

  – EchoService**Stub.java**

    Integra el stub del cliente (proxy)

Desarrollo de aplicaciones CORBA

- **Escribiendo la aplicación:**

  Los ficheros a realizar o modificar son:

  - EchoService**ServerImpl.java**
    - **SERVANT: codigo de aplicación**
    - Proporciona implementación a los métodos que define la interfaz.
    - Es subclase de **EchoPOA**
      - Esto colapsa el mecanismo de herencia en la implementación
  - **Server_AOM.java**
    - **CODIGO CORBA**
    - Contiene el código CORBA del servidor
  - EchoService**ClientImpl.java**
    - **CODIGO CORBA**
    - Implementa el código del cliente

Desarrollo de aplicaciones CORBA

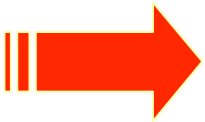
- **Server_AOM.java**
  - **main** (idéntico para cliente y servidor):
    - Define como propiedades las clases del entorno CORBA a utilizar y las del servidor de nombres (si se utiliza):
      - `props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");`
      - `props.put("ooc.orb.service.NameService"`
        `"corbaloc::localhost:1111/NameService");`
    - Llama al método **run** con el código propio de cliente o servidor.
  - **run**:
    - Buscar el POA (Adaptador de Objetos Portable) para conectarse al bus ORB
    - Obtener una referencia IOR al POA manager
    - Salvar la referencia IOR en un fichero y **activarlo implícitamente**
    - Activar al POA manager.

Desarrollo de aplicaciones CORBA

- **EchoServiceClientImpl.java**
  - **main** (idéntico para cliente y servidor):
  - **run**:
    - Obtiene una referencia al objeto "echo“ a partir de un fichero con el IOR en forma de string
    - Narrowing del IOR a tipo Echo
    - Bucle principal del cliente:
      - Leer de teclado
      - invocar el servidor
      - imprimir por pantalla

La arquitectura CORBA

- **Índice**
  - La arquitectura CORBA
  - Desarrollo de aplicaciones CORBA
    - Definición y procesamiento de la interfaz IDL
    - Factorías de objetos
    - El problema de la herencia múltiple
  - El servicio de nombres



Desarrollo de aplicaciones CORBA

- **Factorías de objetos**
  - Una **factoría de objetos** es un objeto que crea **objetos producto** y proporciona acceso a los mismos. Es un punto focal para clientes.
    - *Ejemplo*: gestor de un grupo dinámico de robots.
  - La referencia a una factoría de objetos puede ser publicada en una ubicación bien conocida (servicio de nombres). Los clientes saben que necesitan obtener la referencia a la factoría con el fin de obtener referencias a los *productos* de esa factoría (no publicados en el servicio de nombres).
  - Motivación**:**
    - **Seguridad**: a un cliente la factoría le requiere proporcionar información de seguridad antes de poder acceder un objeto
    - **Balanceo de carga**: mantener un pool de objetos idénticos y repartir la carga.
    - **Polimorfismo**: la factoría maneja punteros a implementaciones diferentes de un mismo interfaz.

Desarrollo de aplicaciones CORBA

- Factorías de objetos

```
// IDL
interface Product
{
      void destroy();
};

interface Factory
{
       Product createProduct();
};
```

Desarrollo de aplicaciones CORBA

- Factorías de objetos

```
// Java
public class Product_impl extends ProductPOA {
    public void destroy() {
        byte[] id = _default_POA().servant_to_id(this);
        _default_POA().deactivate_object(id);
    }
}
```

```
// Java
public class Factory_impl extends FactoryPOA {
    public Product createProduct() {
        Product_impl result = new Product_impl(orb_);
        org.omg.PortableServer.POA poa = ... // Get servant's POA
        byte[] id = ... // Assign an ID
        poa.activate_object_with_id(id, result);
        return result._this(orb_);
    }
}
```

La arquitectura CORBA

- **Índice**
  - La arquitectura CORBA
  - Desarrollo de aplicaciones CORBA
    - Definición y procesamiento de la interfaz IDL
    - Factorías de objetos
    - El problema de la herencia múltiple
  - El servicio de nombres

Desarrollo de aplicaciones CORBA

- Herencia múltiple de Interfaces
  - Interfaz IDL

```
// IDL
interface A
{
    void op_a();
};

interface B
{
   void op_b();
};

interface I : A, B
{
   void op_i();
};
```

Desarrollo de aplicaciones CORBA

- Herencia múltiple de Interfaces
  - Implementación sin herencia

```
// Java
public class I_impl extends IPOA
{
    public void op_a() { ... }

    public void op_b() { ... }

    public void op_i()  { ... }

}
```

Colapsa el mecanismo de la herencia

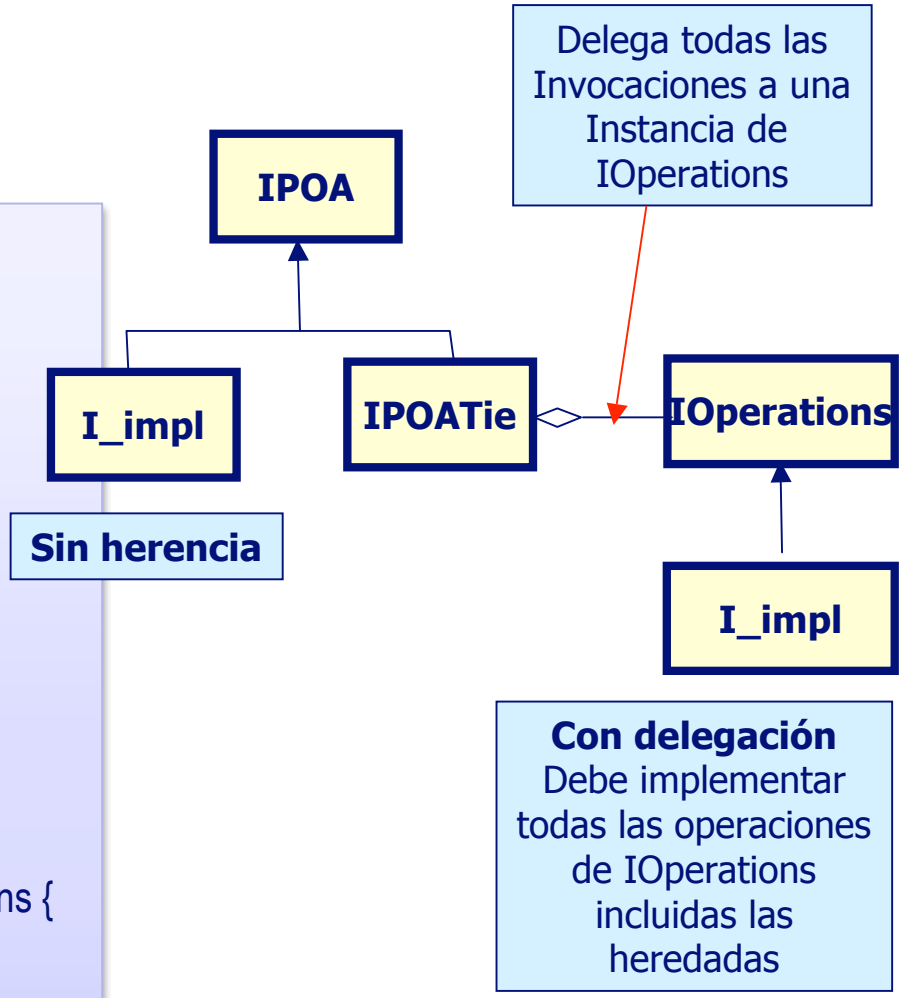
Desarrollo de aplicaciones CORBA

- Herencia múltiple de Interfaces
  - Implementación con delegación

```
// Java
public class A_impl implements AOperations {
    public void op_a() { ...}
}

public class B_impl implements BOperations {
   public void op_b() { ...}
}

public class I_impl extends B_impl implements IOperations {
     public void op_a() { ...}
     public void op_i() { ...}
}
```



La arquitectura CORBA

- **Índice**
  - La arquitectura CORBA
  - Desarrollo de aplicaciones CORBA
    - Definición y procesamiento de la interfaz IDL
    - Factorías de objetos
    - El problema de la herencia múltiple
  - El servicio de nombres

El servicio de nombres CORBA

- **El servicio de nombres**
  - El servicio de nombres CORBA mantiene un conjunto de referencias a objetos organizadas de manera jerárquica, similar a un sistema de ficheros:
    - fichero → **binding**
    - Directorio → **naming context**
  - Un *binding* es un es el nombre de un objeto y su tipo, tal como se define en el modulo **`CosNaming`**
  - Tiene una especificación IDL, como cualquier objeto CORBA.

El servicio de nombres CORBA

- Bindings: definición

```
// IDL
typedef string Istring;

struct NameComponent
{ Istring id;
  Istring kind;
};

typedef sequence<NameComponent> Name;

enum BindingType
{ nobject,
  ncontext
};
```

```
struct Binding
{ Name binding_name;
  BindingType binding_type;
};
```

El servicio de nombres CORBA

- Métodos para registrar nuevos *bindings*

```
// IDL
void bind (in Name n, in Object obj)
        raises(NotFound, CannotProceed, InvalidName, AlreadyBound);

void bind_context(in Name n, in NamingContext nc)
        raises(NotFound, CannotProceed, InvalidName, AlreadyBound);

NamingContext new_context();

NamingContext bind_new_context(in Name n)
        raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
```

El servicio de nombres CORBA

- Métodos para modificar / eliminar *bindings*

```
// IDL
void rebind(in Name n, in Object obj)
        raises(NotFound, CannotProceed, InvalidName);
void rebind_context(in Name n, in NamingContext nc)
        raises(NotFound, CannotProceed, InvalidName);


// IDL
void unbind(in Name n)
       raises(NotFound, CannotProceed, InvalidName);
```

El servicio de nombres CORBA

- Métodos para resolver / listar *bindings*

```
// IDL
Object resolve(in Name n)
        raises(NotFound, CannotProceed, InvalidName);
Object resolve_str(in StringName n)
        raises(NotFound, CannotProceed, InvalidName);


// IDL
typedef sequence<Binding> BindingList;
void list(in unsigned long how_many, out BindingList bl, out BindingIterator bi);
```

El servicio de nombres CORBA

- Métodos para iterar sobre *bindings*

```
// IDL
interface BindingIterator
{
    boolean next_one(out Binding b);
    boolean next_n(in unsigned long how_many, out BindingList bl);
    void destroy();
};
```

El servicio de nombres CORBA

- **El servidor**

```
// Crear una instancia del objeto
  Camara_impl camaraImpl = new Camara_impl();
  Camara camara = camaraImpl._this(orb);
// Registrarlo en el Name Service
  org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
  NamingContext ncRef = NamingContextHelper.narrow(objRef);
  NameComponent nc = new NameComponent (“Camara", "");
  NameComponent path[] = {nc};
  ncRef.rebind(path,camara);
```

- **El cliente**

```
org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
NamingContext ncRef = NamingContextHelper.narrow (objRef);
NameComponent nc = new NameComponent (“Camara", "");
NameComponent path[] = {nc};
Camara camara = CamaraHelper.narrow(ncRef.resolve(path));
```

El servicio de nombres CORBA

- Un ejemplo

```
// Java
java.util.Properties props = System.getProperties();
p r o p s . p u t ( " o r g . o m g . C O R B A . O R B C l a s s " ,
    "com.ooc.CORBA.ORB");
props.put("org.omg.CORBA.ORBSingletonClass",
            "com.ooc.CORBA.ORBSingleton");
try {
     orb = ORB.init(args, props);
     org.omg.CORBA.Object poaObj = null;
    try {
        poaObj = orb.resolve_initial_references("RootPOA");
      } catch(org.omg.CORBA.ORBPackage.InvalidName ex) {
         throw new RuntimeException();
      }
      POA rootPOA = POAHelper.narrow(poaObj);
      POAManager manager = rootPOA.the_POAManager();
```

```
org.omg.CORBA.Object obj = null;
try {
     obj = orb.resolve_initial_references("NameService");
   } catch(org.omg.CORBA.ORBPackage.InvalidName ex) {
     throw new RuntimeException();
   }
if(obj == null) {
   throw new RuntimeException();
}

NamingContextExt nc = null;
try {
    nc = NamingContextExtHelper.narrow(obj);
    } catch(org.omg.CORBA.BAD_PARAM ex) {
    throw new RuntimeException();
}
```

El servicio de nombres CORBA

```
// Java
Named_impl implA = new Named_impl();
Named_impl implA1 = new Named_impl();
Named_impl implA2 = new Named_impl();
Named_impl implA3 = new Named_impl();
Named_impl implB = new Named_impl();
Named_impl implC = new Named_impl();
Named a = implA._this(orb);
Named a1 = implA1._this(orb);
Named a2 = implA2._this(orb);
Named a3 = implA3._this(orb);
Named b = implB._this(orb);
Named c = implC._this(orb);
```

Crea objetos y los activa implícitamente

```
try {
    NameComponent[] nc1Name = new NameComponent[1];
    nc1Name[0] = new NameComponent();
    nc1Name[0].id = "nc1";
    nc1Name[0].kind = "";
NamingContext nc1 = nc.bind_new_context(nc1Name);

    NameComponent[] nc2Name = new NameComponent[2];
    nc2Name[0] = new NameComponent();
    nc2Name[0].id = "nc1";
    nc2Name[0].kind = "";
    nc2Name[1] = new NameComponent();
    nc2Name[1].id = "nc2";
    nc2Name[1].kind = "";
    NamingContext nc2 = nc.bind_new_context(nc2Name);
```

Crea contexto “nc1” en /

Crea contexto “nc2” en /nc1

El servicio de nombres CORBA

```
NameComponent[] aName = new NameComponent[1];
aName[0] = new NameComponent();
aName[0].id = "a";
aName[0].kind = "";
nc.bind(aName, a);

NameComponent[] bName = new NameComponent[2];
bName[0] = new NameComponent();
bName[0].id = "nc1";
bName[0].kind = "";
bName[1] = new NameComponent();
bName[1].id = "b";
bName[1].kind = "";
nc.bind(bName, b);
```

Inserta nombre “a” en /

Inserta nombre b en /nc1

El Servicio de Nombres de Orbacus

- **Como activar el Servicio de Nombres Orbacus?**
  - Incluir en el proyecto los jars:
    - OB.jar
    - OBNaming.jar
    - OBUtil.jar
  - Ejecutar la clase (crear una configuración de ejecución):
    - **com.ooc.cosNaming.Server**
    - Parámetro de ejecución: **-OAport 1111**
- **Servicio de Nombres**
  - Utilizar el siguiente parámetro del VM:
    - **-Dooc.orb.service.NameService=corbaloc::localhost:1111/NameService**
- **Consola del Servicio de Nombres Orbacus**
  - Ejecutar la clase (crear una configuración de ejecución):
    - **com.ooc.cosNamingConsole.Main**
    - Parámetro de VM:
      Dooc.orb.service.NameService=corbaloc::localhost:1111/NameService