Desarrollo de Aplicaciones Distribuidas

AUTORES:
Alejandro Calderón Mateos
Javier García Blas
David Expósito Singh
Laura Prada Camacho

Departamento de Informática
Universidad Carlos III de Madrid
Julio de 2012

CORBA

Contenidos

1. Introducción:
   1. Paradigma de *Object Request Broker*
2. CORBA en Java
   1. Introducción
   2. Arquitectura
   3. Ejemplo de aplicación
      1. Interfaz y despliegue
      2. Empleo de IOR

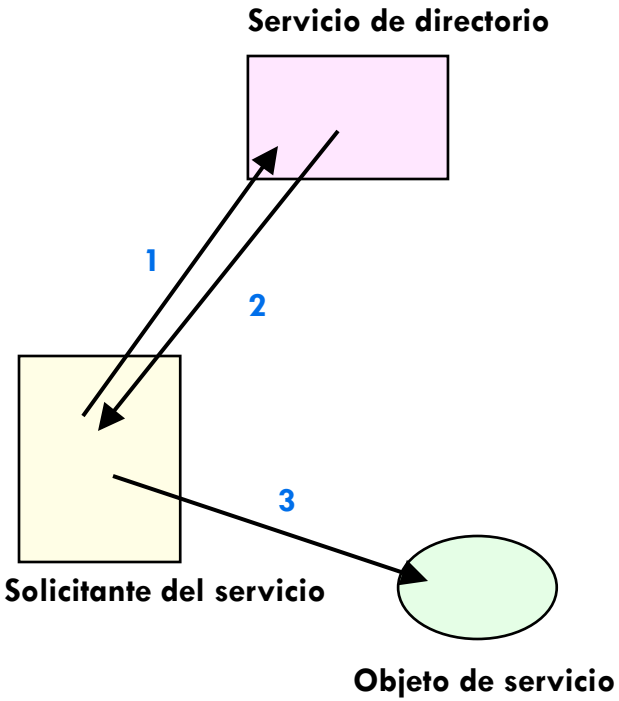
Contenidos

1. **Introducción:**
   1. **Paradigma de *Object Request Broker***
2. CORBA en Java
   1. Introducción
   2. Arquitectura
   3. Ejemplo de aplicación
      1. Interfaz y despliegue
      2. Empleo de IOR

Paradigmas de Servicios de red, ORB y agentes móviles

Espacio de objetos, aplicaciones colaborativas
Servicios de red, *object request broker*, agentes móviles
procedimientos remotos, métodos remotos
Cliente-servidor, *peer-to-peer*
Paso de mensajes

Paradigma de servicios de red



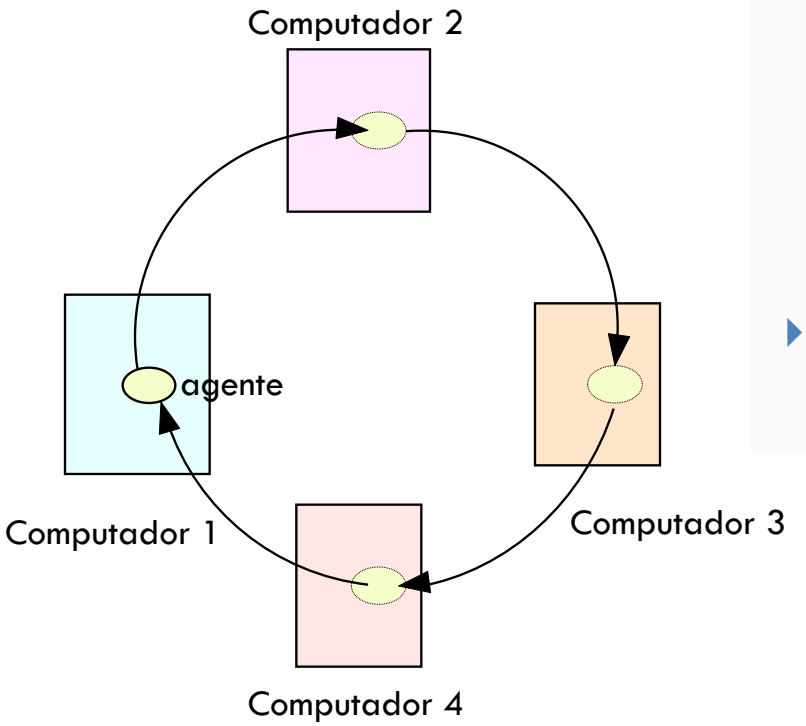
- Servicio de directorio: proporcionan la referencia a los servicios disponibles

- Pasos:
  1. El proceso solicitante contacta con el servicio de directorio
  2. El servicio de directorio devuelve la referencia al servicio solicitado
  3. Usando la referencia, el proceso solicitante interactúa con el servicio

Paradigma de servicios de red

- Extensión del paradigma de invocación de métodos remotos
- Transparencia de localización: nivel de abstracción extra
- Ejemplos:
  - Tecnología *Jini* de Java
  - Protocolo SOAP lo aplica para servicios accesibles en la Web

Paradigma de agentes móviles



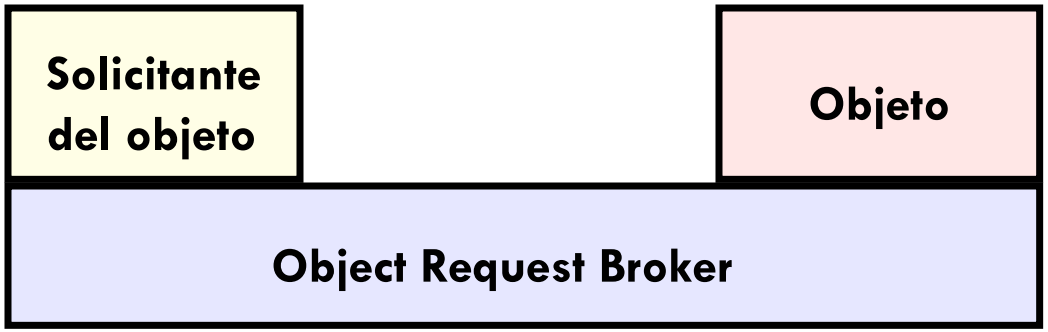
- Agente móvil:
  programa u objeto transportable.
  - Un agente se lanza desde un ordenador
  - Viaja de forma automática de acuerdo con un itinerario
- Accede a los recursos o servicios de cada sistema que visita

Paradigma de agentes móviles

- Ejemplos:
  - Concordia system de Mitsubishi Electric ITA.
  - Aglet system de IBM.
  - JADE
- Sistemas experimentales:
  - D'agent.
  - Proyecto Tacoma.

Paradigma basado en *Object Request Broker*

- El ORB funciona como una capa middleware.
- El ORB redirige las peticiones al objeto apropiado que proporciona el servicio solicitado.
- Extensión a los paradigmas a RMI y servicios de red:
  - Instanciación de clases y objetos



Paradigma basado en *Object Request Broker*

- El ORB actúa como mediador de objetos heterogéneos

- Ejemplos:
  - CORBA (*Common Object Request Broker Architecture*)
    - Java CORBA
    - *Visibroker* de Inspire.
    - *IONA* de Orbix y *TAO* de *Objet Computing, Inc.*
  - *Microsoft* COM, DCOM.
  - *Java Beans* y *Enterprise Java Beans.*

Contenidos

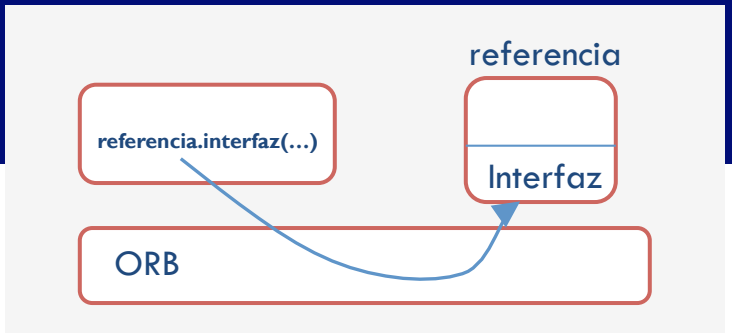
1. Introducción:
   1. Paradigma de *Object Request Broker*
2. **CORBA en Java**
   1. **Introducción**
   2. Arquitectura
   3. Ejemplo de aplicación
      1. Interfaz y despliegue
      2. Empleo de IOR

Terminología

- ¿Qué es CORBA?
  - CORBA o *Common Object Request Broker Architecture*, es una arquitectura estándar para sistemas de objetos distribuidos.
  - Permite interoperar a una colección de objetos distribuida y heterogénea.

- ¿Qué es el OMG?
  - OMG o *Object Management Group* es el responsable de la definición de CORBA.
  - Es un compromiso de cerca de 700 empresas y organizaciones, entre ellas los principales vendedores y desarrolladores de tecnologías de objetos distribuidos.

http://www.omg.org/



Terminología

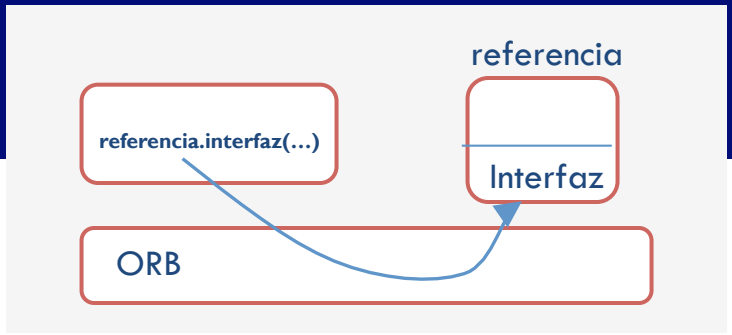
- La arquitectura CORBA
  - CORBA define una arquitectura para objetos distribuidos.
  - El paradigma básico de CORBA es en el que se hace una petición para obtener un servicio de un objeto distribuido. Todo lo demás definido por el OMG está descrito en términos de este paradigma básico.

- El uso de interfaces
  - Los servicios que proporciona un objeto son dados por su interfaz.
  - Las interfaces se definen en el IDL (*Interface Definition Language*) dado por el OMG.
  - Los objetos distribuidos están identificados por referencias a objetos, las cuales se describen mediante los interfaces IDL.

http://java.sun.com/developer/onlineTraining/corba/corba.html

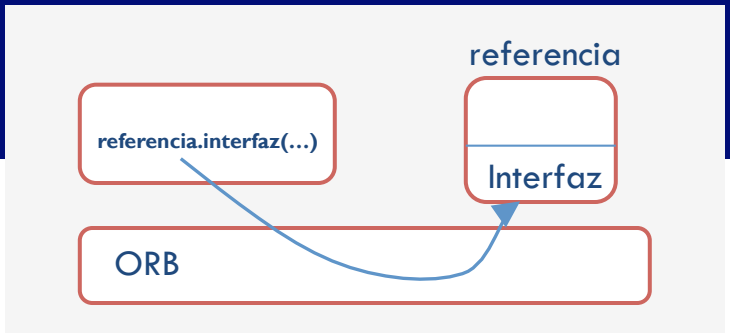
CORBA: Caso de estudio

- EADS CASA CN-235 Persuader



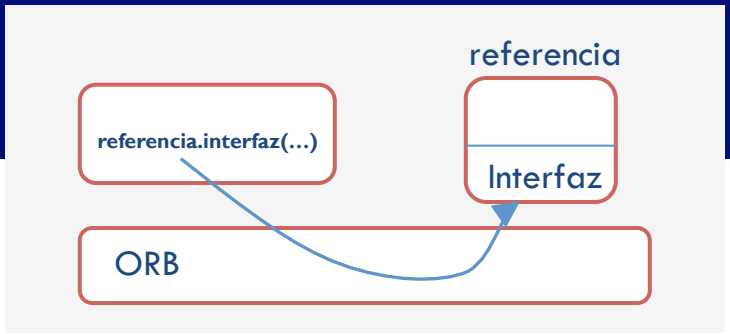
Terminología

- El ORB
  - El ORB es el servicio distribuido que implementa la petición al objeto remoto.
  - Localiza el objeto remoto en la red, le comunica la petición, espera a los resultados y cuando están disponibles se los devuelve al cliente.
  - El ORB implementa transparencia de localización. Se usa exactamente el mismo mecanismo de peticiones, sin importar donde esté localizado.
  - El ORB implementa independencia del lenguaje de programación para las peticiones. El cliente que lanza la petición se puede escribir en uno de los diferentes lenguajes que aceptan objetos CORBA. El ORB hace las conversiones necesarias entre lenguajes de programación.



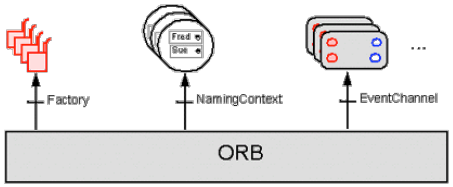
Terminología

- El IIOP
  - Uno de los objetivos de la especificación CORBA es que las implementaciones de los clientes y servidores sean portables:
    - La especificación CORBA define un API para los clientes de objetos distribuidos así como un API para la implementación de un objeto CORBA.
    - La realidad de los productos CORBA del mercado de hoy es que los clientes CORBA son portables, pero las implementaciones de los objetos necesitan un poco de esfuerzo para portarse de un producto CORBA a otro.
  - …



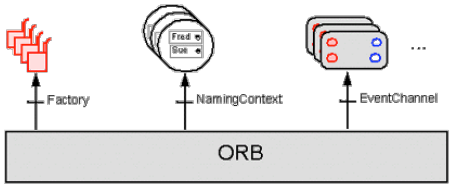
Terminología

- El IIOP
  - …
  - CORBA 2.0 añadió la interoperabilidad como un objetivo en la especificación:
    - En particular, CORBA 2.0 define un protocolo de red llamado IIOP (*Internet Inter-ORB Protocol*), que permite a los clientes usar productos CORBA de cualquier desarrollador que se comuniquen con objetos CORBA de cualquier otro desarrollador.
    - IIOP trabaja sobre Internet, o más exactamente, sobre cualquier implementación de TCP/IP.
    - IIOP se usa en otros sistema que incluso no intentan proporcionar el API de CORBA. Por ejemplo, hay una versión de Java RMI que utiliza IIOP como protocolo de transporte.



Terminología

- Los COS
  - Otra parte importante del estándar CORBA es la definición de un conjunto de servicios distribuidos para facilitar la integración e interoperabilidad de los objetos distribuidos.
  - Los servicios de CORBA, conocidos como *CORBA Services* o COS, se encuentran definidos en la parte superior del ORB.
  - Están definidos como objetos CORBA estándares con interfaces IDL, algunas veces llamados *Object Services*.



COS más populares

Servicio	Descripción
Ciclo de vida del objeto	Define como los objetos CORBA se crean, eliminan, mueven y copian.
Nominación	Define como los objetos CORBA pueden tener nombres simbólicos amigables.
Eventos	Desacopla la comunicación entre objetos distribuidos.
Relaciones	Proporciona un red arbitraria de relaciones entre objetos CORBA.
Externalización	Coordina la transformación entre objetos CORBA a y desde el medio externo.
Transacciones	Coordina accesos atómicos a objetos CORBA.
Control de concurrencia	Proporciona un servicio de bloqueo para objetos CORBA que asegura accesos serializables.
Propiedad	Soporta la asociación de pares nombre-valor con objetos CORBA.
Negociación	Soporta la búsqueda de objetos CORBA basados en propiedades que describen el servicio ofrecido por el objeto.
Consultas	Soporta las consultas a objetos.

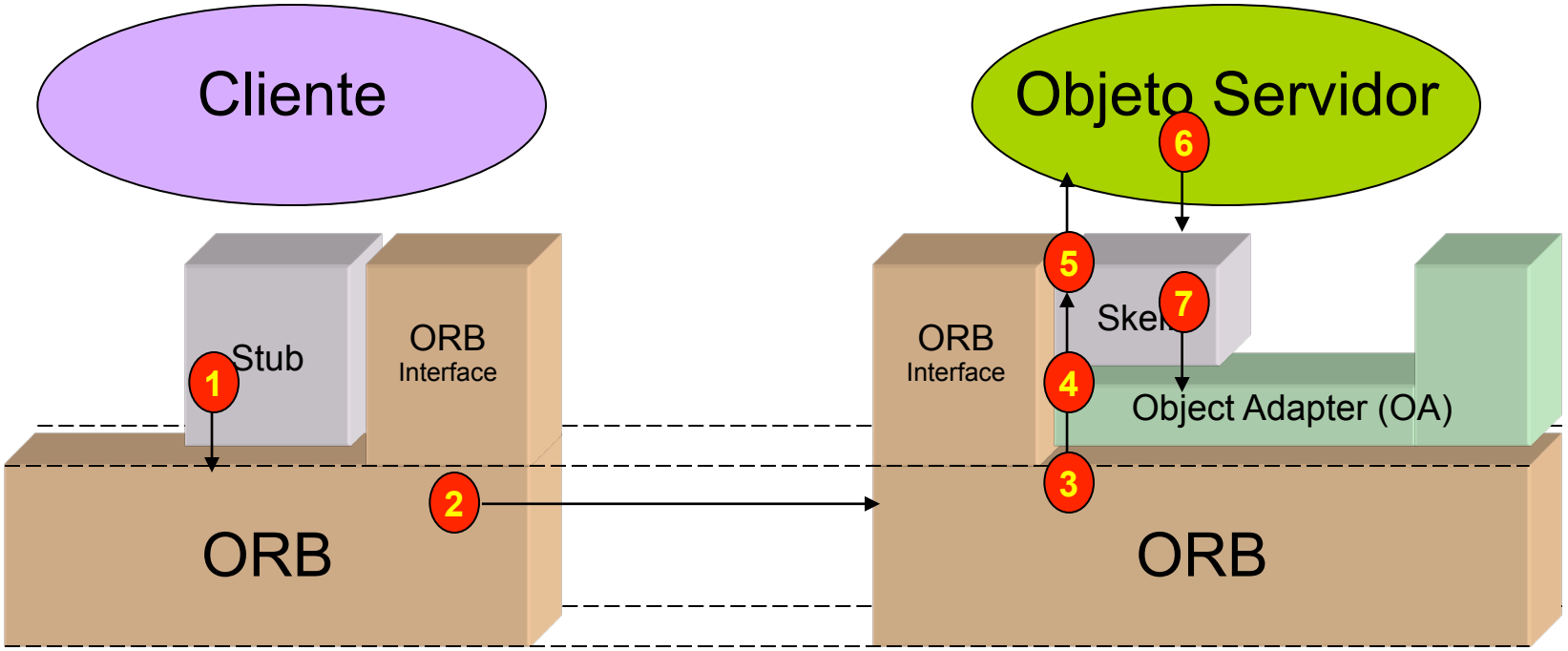
Productos CORBA

ORB	Descripción
Java 2 ORB	Viene con el Java 2 SDK de Sun. **No** implementa algunas funcionalidades.
VisiBroker for Java	ORB para Java muy popular de Inprise Corporation. VisiBroker también lo podemos encontrar empotrado en otros productos, como en el Netscape Communicator.
OrbixWeb	ORB para Java de Iona Technologies.
WebSphere	Un servidor de aplicaciones con un ORB de IBM.
ORBit, Mico, TAO, OmniORB2, ...	Existen implementaciones de CORBA para varios lenguajes que se facilitan bajo licencia de software libre (o shareware): • http://patriot.net/~tvalesky/freecorba.html • http://www.omg.org/technology/corba/corbadownloads.htm

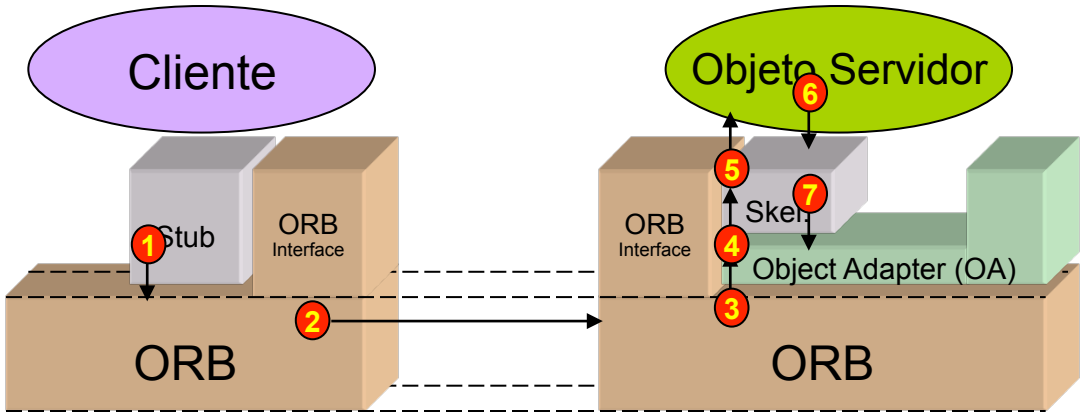
Contenidos

1. Introducción:
   1. Paradigma de *Object Request Broker*
2. **CORBA en Java**
   1. Introducción
   2. **Arquitectura**
   3. Ejemplo de aplicación
      1. Interfaz y despliegue
      2. Empleo de IOR

Comunicación mediante CORBA



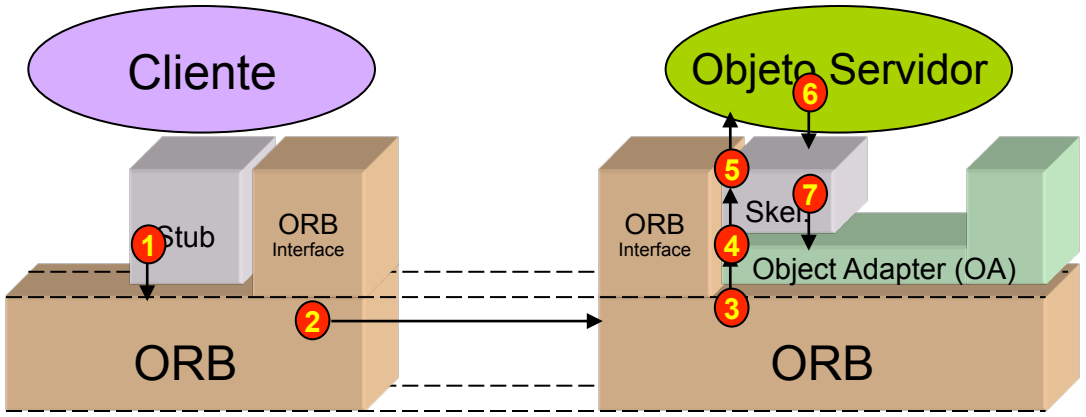
Comunicación mediante CORBA



1. El cliente realiza una petición usando *stubs* estáticos (previamente compilados) y la dirige a su ORB.
2. El ORB cliente transmite las peticiones al ORB enlazado con el servidor.
3. El ORB del servidor redirige la petición al *adaptador de objetos* que ha creado el objeto destino.
4. El adaptador de objetos dirige la petición al servidor que implementa el objeto destino vía el *skeleton*.

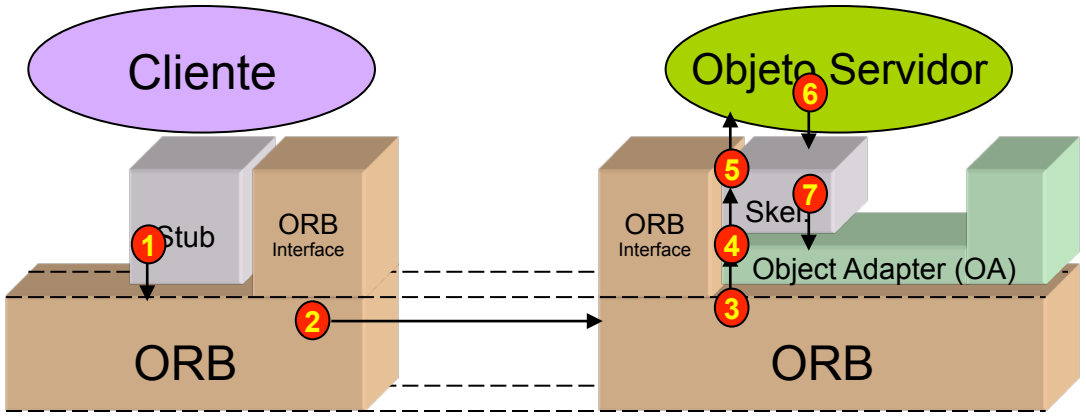
10. El servidor devuelve su respuesta al *skeleton* el cual, devuelve el resultado siguiendo los mismos pasos.

Funciones del ORB



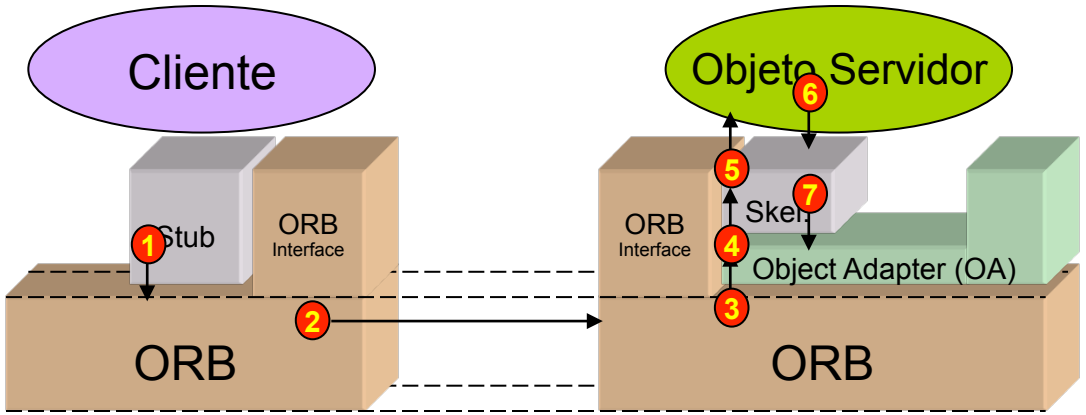
1. Localización de objetos:
   El cliente desconoce el computador donde se encuentra el objeto remoto.
2. Comunicación independiente entre cliente y servidor:
   Comunicar de forma independiente de protocolos de comunicación o características de implementación (lenguaje, sistema operativo, ...)

Funciones del OA



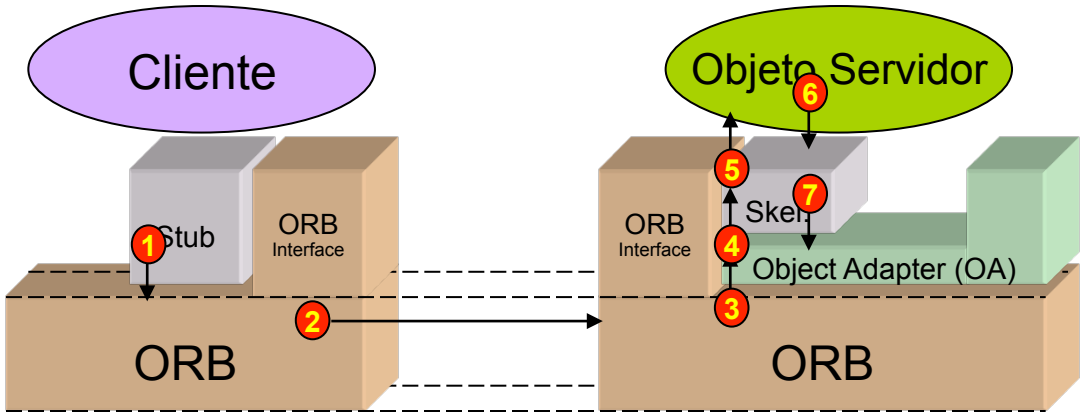
1. Un adaptador de objeto (OA) es un objeto que adapta la interfaz de un objeto a una interfaz esperada por un usuario:
   1. Crean referencias de objetos.
   2. Aseguran que cada objeto destino esté encarnado en un sirviente (entidad que implementa uno o más objetos CORBA)
   3. Reciben las peticiones emitidas por el ORB del servidor y las redirigen a los sirvientes que encarnan a los objetos destino.

Tipos de petición



1. Una petición es una invocación de una operación de un objeto CORBA realizada por un cliente:
   1. Petición síncrona. El cliente se bloquea esperando la respuesta. Idénticas a las llamadas a procedimientos remotos.
   2. Petición de sentido único (*oneway*). El cliente no espera respuesta.
   3. También existen peticiones asíncronas.

Referencias a objetos



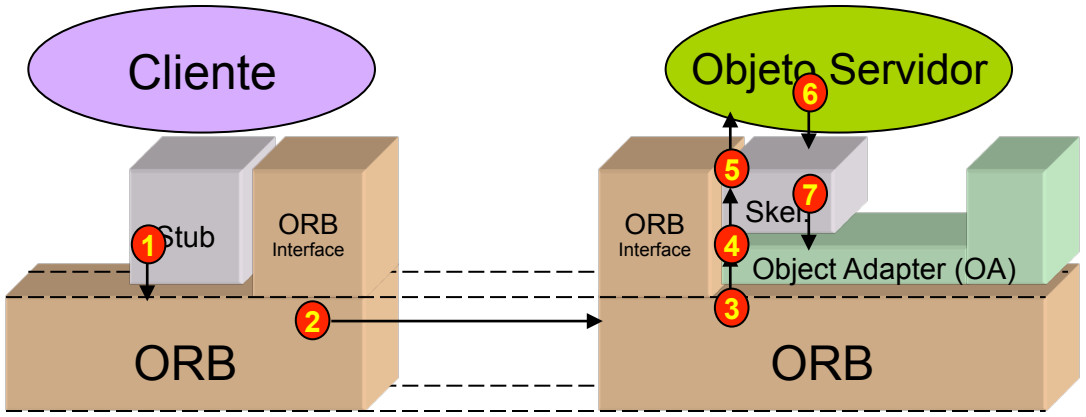
1. Los objetos de una aplicación CORBA se encuentran identificados por medio de una referencia única:
   1. Esta referencia es generada al activar un objeto en el Adaptador de Objetos.
   2. Por medio de esta referencia el ORB es capaz de localizar el computador remoto y el adaptador de objetos donde se encuentra. Éste último es capaz de identificar el objeto concreto dentro del adaptador.
   3. …

Referencias a objetos

```
IOR:
010000000f00000049444c3a4375656e74613a312e30
000002000000000000003000000001010000160000000
7175696e6f2e64617473692e66692e75706d2e657300
41040c000000424f418a640965000009f40301000000
2400000001000000010000000100000014000000100
000001000100000000009010100000000000
```

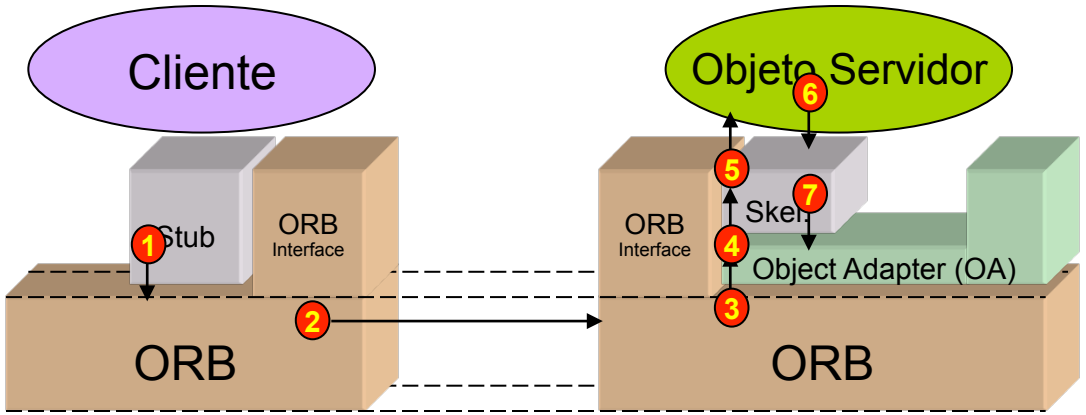
- Protocolo *Interoperable Object Reference* (IOR)
- Codifica la siguiente información:
  1. Tipo de objeto.
  2. Servidor del objeto.
  3. Puerto asociado al servidor del objeto.
  4. Clave del objeto.

Funciones del ORB (2)



- Para que un cliente envíe un mensaje a un objeto necesita tener una referencia de dicho objeto.
- Cuando un cliente llama a una operación, el ORB:
  - Localiza al objeto destino.
  - Activa a la aplicación servidor, si no está activa.
  - Transmite los argumentos.
  - Activa un sirviente para el objeto si es necesario.
  - Espera hasta que se complete la operación.
  - Devuelve cualquier parámetro `out` e `inout` al cliente.
  - Devuelve una excepción cuando falla la llamada.

Funciones del ORB (2)



- Características de las invocaciones:
  - Transparencia de la localización.
  - Transparencia del servidor.
  - Independencia del lenguaje.
  - Independencia de la implementación.
  - Independencia de la arquitectura.
  - Independencia del sistema operativo.
  - Independencia del protocolo.
  - Independencia del nivel transporte.

Contenidos

1. Introducción:
   1. Paradigma de *Object Request Broker*
2. **CORBA en Java**
   1. Introducción
   2. Arquitectura
   3. **Ejemplo de aplicación**
      1. **Interfaz y despliegue**
      2. Empleo de IOR

Hola.idl

- Descripción de los métodos que un objeto proporciona al resto del entorno
- Uso de IDL (Lenguaje de Definición de Interfaz)

```
module HolaApp
{
   interface Hola
   {
      string saluda();
   };
};
```

Preprocesado del IDL a Java

```
user@guernika# idlj -fall Hola.idl
```

- Genera los siguientes archivos Java en el directorio **`HolaApp`**:
  - HolaOperations.java: Interfaz de operaciones.
  - Hola.java: Interfaz IDL en Java.
  - HolaHelper.java: Funcionalidad auxiliar para conversiones.
  - HolaHolder.java: Referencia al objeto que implementa la interfaz.
  - HolaPOA.java: Clase abstracta para el esqueleto y adaptador de objetos.
  - _HolaStub.java: Funcionalidad CORBA en el cliente.

Preprocesado del IDL a Java

```
user@guernika# idlj -fall Hola.idl
```

- NO genera (y el programador ha de escribir):
  - HolaImpl.java: Implementación del servicio.
  - Servidor.java: Método principal del servidor CORBA.
  - Cliente.java: Método principal del cliente CORBA.

Cliente.java (1/2)

```
import HolaApp.*;              // El paquete que contienes nuestros stubs
import org.omg.CosNaming.*; // HolaClient usará el servicio de nombrado
import org.omg.CORBA.*;      // HolaClient usará las clases de CORBA

public class Cliente
{
   public static void main(String args[])
   {
     try {
       // Crea e inicializa el ORB
       ORB orb = ORB.init(args, null);

       // Obtiene el contexto de nombrado raiz
       org.omg.CORBA.Object objRef =
                           orb.resolve_initial_references("NameService");
       NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

Cliente.java (2/2)

```
        // Resuelve la referencia al objeto en el nombrado
        NameComponent nc = new NameComponent("Hola", "");
        NameComponent path[] = { nc };
        Hola HolaRef = HolaHelper.narrow(ncRef.resolve(path));

        // Llama al objeto servidor Hola e imprime el resultado
        String Hola = HolaRef.saluda();
        System.out.println(Hola);
      }
      catch(Exception e)
      {
         System.out.println("ERROR : " + e);
         e.printStackTrace(System.out);
      }
   }
}
```

HolaImpl.java

```
import HolaApp.*;

public class HolaImpl extends HolaPOA
{
    public String saluda()
    {
          System.out.println("Hola1");
          return "Hola2" ;
    }
}
```

Servidor.java (1/3)

```
import HolaApp.*;
import java.util.Properties;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;

public class Servidor {
  public static void main(String args[]) {
    try {
        // Iniciación de ORB
        ORB orb = ORB.init(args,null);

        // Raíz POAA
   POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
```

Servidor.java (2/3)

```
// Creación de un sirviente y su registro en el ORB
HolaImpl HolaServant = new HolaImpl();
rootPOA.activate_object(HolaServant);

Hola HolaRef = HolaHelper.narrow(
                    rootPOA.servant_to_reference(HolaServant));


// Asocia la referencia con el servicio
org.omg.CORBA.Object refCN = orb.resolve_initial_references("NameService");

NamingContext contextoNombrado=NamingContextHelper.narrow(refCN);

NameComponent nombre = new NameComponent("Hola", "");
NameComponent camino[] = { nombre };
contextoNombrado.bind(camino, HolaRef);
```

Servidor.java (3/3)

```
        // Activacion del rootpoa
        rootPOA.the_POAManager().activate();

        // Activacióna del ORB
        orb.run();

      }
      catch (Exception e)
      {
          System.err.println("Error: " + e);
          e.printStackTrace(System.err);
      }

   }
}
```

Despliegue del ejemplo

guernika.lab.inf.uc3m.es

```
user@guernika # find . –type f
./Hola.idl
./HolaApp/HolaPOA.java
./HolaApp/_HolaStub.java
./HolaApp/HolaHolder.java
./HolaApp/HolaHelper.java
./HolaApp/Hola.java
./HolaApp/HolaOperations.java
./Servidor.java
./Cliente.java
./HolaImpl.java
```

Compilación del ejemplo

guernika.lab.inf.uc3m.es

```
user@guernika # javac1.6 -g  Cliente.java    \
                                    HolaApp/*.java

user@guernika # javac1.6 -g  Servidor.java  \
                                    HolaImpl.java HolaApp/*.java
```

Ejecución del ejemplo

guernika.lab.inf.uc3m.es

```
user@guernika # orbd1.6  –ORBInitialPort <número de puerto>

user@guernika # java1.6  Servidor    \
                                -ORBInitialPort    <número de puerto> \
                                -ORBInitialHost    <host>    &

user@guernika # java1.6  Cliente    \
                                -ORBInitialPort    <número de puerto> \
                                -ORBInitialHost    <host>    &
```

Contenidos

1. Introducción:
   1. Paradigma de *Object Request Broker*
2. **CORBA en Java**
   1. Introducción
   2. Arquitectura
   3. Ejemplo de aplicación
      1. Interfaz y despliegue
      2. **Empleo de IOR**

IORServidor.java (1/3)

```
import HolaApp.*;
import java.util.Properties;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;

public class IORServidor {
  public static void main(String args[]) {
    try {
        // Iniciación de ORB
        ORB orb = ORB.init(args,null);

        // Raíz POAA
   POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
```

IORServidor.java (2/3)

```
// Creación de un sirviente y su registro en el ORB
HolaImpl HolaServant = new HolaImpl();
rootPOA.activate_object(HolaServant);

Hola HolaRef = HolaHelper.narrow(
                    rootPOA.servant_to_reference(HolaServant));


// Asocia la referencia con el servicio
org.omg.CORBA.Object refCN = orb.resolve_initial_references("NameService");

NamingContext contextoNombrado=NamingContextHelper.narrow(refCN);

NameComponent nombre = new NameComponent("Hola", "");
NameComponent camino[] = { nombre };
contextoNombrado.bind(camino, HolaRef);
```

IORServidor.java (3/3)

```
        // Get a stringified reference to the object
        String sor = orb.object_to_string(HolaRef);
        System.out.println("HolaImpl IOR: " + sor);

        // Activacion del rootpoa
        rootPOA.the_POAManager().activate();

        // Activacióna del ORB
        orb.run();
      }
      catch (Exception e)
      {
          System.err.println("Error: " + e);
          e.printStackTrace(System.err);
      }
    }
}
```


http://docstore.mik.ua/orelly/java-ent/jenut/ch04_04.htm

IORCliente.java (1/2)

```
import HolaApp.*;
import org.omg.CORBA.*;

public class IORCliente
{
  public static void main(String[] argv)
  {
    // Get the stringified reference from our command-line arguments
    String sor = null;
    if (argv.length > 0) {
         sor = argv[0];
    }
    else {
      System.out.println("You forgot the object reference...");
      System.exit(1);
    }
```


http://docstore.mik.ua/orelly/java-ent/jenut/ch04_04.htm

IORCliente.java (2/2)

```
    try {
      // Obtain ORB reference
      ORB myORB = ORB.init(argv, null);

      // Convert the stringified reference into a live object reference
      org.omg.CORBA.Object objRef = myORB.string_to_object(sor);

      // Narrow the object reference to a IORServant
      Hola server = HolaHelper.narrow(objRef);

      // Invoke some methods on the remote object through the stub
      server.saluda();
    }
    catch (Exception e) {
      System.out.println("Error occurred while initializing server object:");
      e.printStackTrace();
    }
  }
}
```


http://docstore.mik.ua/orelly/java-ent/jenut/ch04_04.htm