Sistemas Operativos Distribuidos

RMI: Invocación de método remoto

- Java RMI
- CORBA

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Modelo de objetos en sistemas distribuidos

- Sistemas distribuidos.
  - Aplicaciones inherentemente distribuidas.
  - Se caracterizan por su complejidad.
- Sistemas orientados a objetos.
  - Más cercanos al lenguaje natural.
  - Facilitan el diseño y la programación.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Objetos-Distribuidos

Características:

- Uso de un *Middleware*: Nivel de abstracción para la comunicación de los objetos distribuidos. Oculta:
  - Localización de objetos.
  - Protocolos de comunicación.
  - Hardware de computadora.
  - Sistemas Operativos.
- Modelo de objetos distribuidos: Describe los aspectos del paradigma de objetos que es aceptado por la tecnología: Herencia, Interfaces, Excepciones, Polimorfismo, ...
- Recogida de basura (*Garbage Collection*): Determina los objetos que no están siendo usados para liberar recursos.

Ventajas respecto a paso de mensajes

- Paso de mensajes:
  - Procesos fuertemente acoplados
  - Paradigma orientado a datos: No adecuado para aplicaciones muy complejas que impliquen un gran número de peticiones y respuestas entremezcladas.
- Paradigma de objetos distribuidos
  - Mayor abstracción
  - Paradigma orientado a acciones:
    - Hace hincapié en la invocación de las operaciones
    - Los datos toman un papel secundario

Ventajas respecto a RPC

- Ventajas derivadas al uso de programación orientada a objetos:
  - Encapsulación
  - Reutilización
  - Modularidad
  - Dinamismo

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Objetos distribuidos

- Minimizar las diferencias de programación entre las invocaciones de métodos remotos y las llamadas a métodos locales
- Ocultar las diferencias existentes:
  - Tratamiento del empaquetamiento de los datos (*marshalling*)
  - Sincronización de los eventos
  - Las diferencias deben quedar ocultas en la **arquitectura**

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Sistemas Operativos Distribuidos

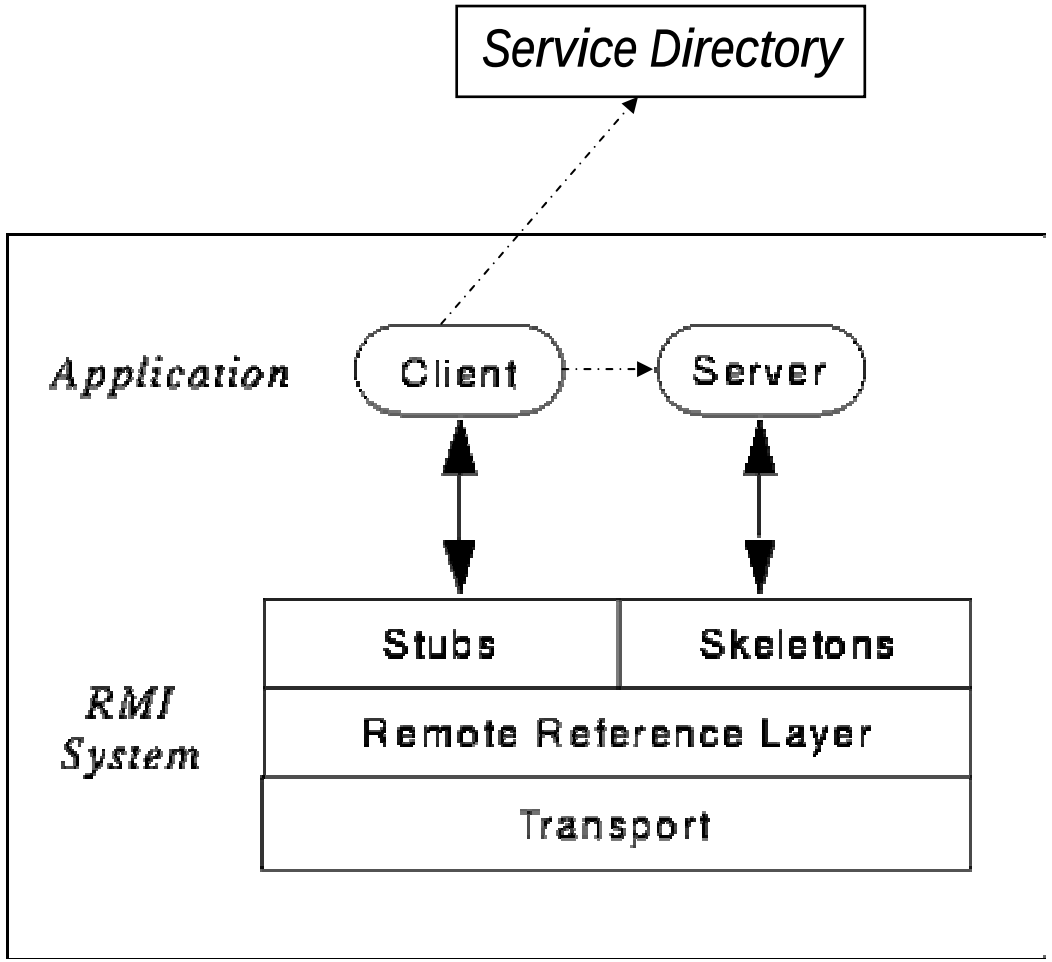
Java RMI

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Java RMI

- Java: *Write Once, Run Anywhere*
- Java RMI extiende el modelo Java para la filosofía *"Run Everywhere"*

Arquitectura de Java RMI



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Arquitectura de Java RMI

- *Nivel de resguardo/esqueleto* (*proxy/skeleton*) que se encarga del aplanamiento (serialización) de los parámetros
  - *proxy*: resguardo local. Cuando un cliente realiza una invocación remota, en realidad hace una invocación de un método del resguardo local.
  - Esqueleto (*skeleton*): recibe las peticiones de los clientes, realiza la invocación del método y devuelve los resultados.
- *Nivel de gestión de referencias remotas*: interpreta y gestiona las referencias a los objetos de servicio remoto
- *Nivel de transporte*: se encarga de las comunicaciones y de establecer las conexiones necesarias. Basada en TCP (orientada a conexión)

Servicio de directorios

- Se pueden utilizar diferentes servicios de directorios para registrar un objeto distribuido
  - Ejemplo: JNDI (Java Naming and Directory Interface)
- El registro RMI, *rmiregistry*, es un servicio de directorios sencillo proporcionado por SDK (Java Software Development Kit)

Java RMI

El soporte para RMI en Java está basado en las interfaces y clases definidas en los paquetes:

- *java.rmi*
- *java.rmi.server*

Características de Java RMI:

- Se basa en una interfaz remota Java (hereda de la clase Java *Remote*).
- Es necesario tratar mayor número de excepciones que en el caso de invocación de métodos locales
  - Errores en la comunicación entre los procesos (fallos de acceso, fallos de conexión)
  - Problemas asociados a la invocación de métodos remotos (no encontrar el objeto, el resguardo o el esqueleto)
  - Los métodos deben especificar la excepción *RemoteException*.

Ejemplo de interfaz remota Java

```
public interface InterfazEj extends
  java.rmi.Remote
{
  public String metodoEj1()
     throws java.rmi.RemoteException;
  public int metodoEj2(float param)
     throws java.rmi.RemoteException;
}
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Java RMI

Localización de objetos remotos:

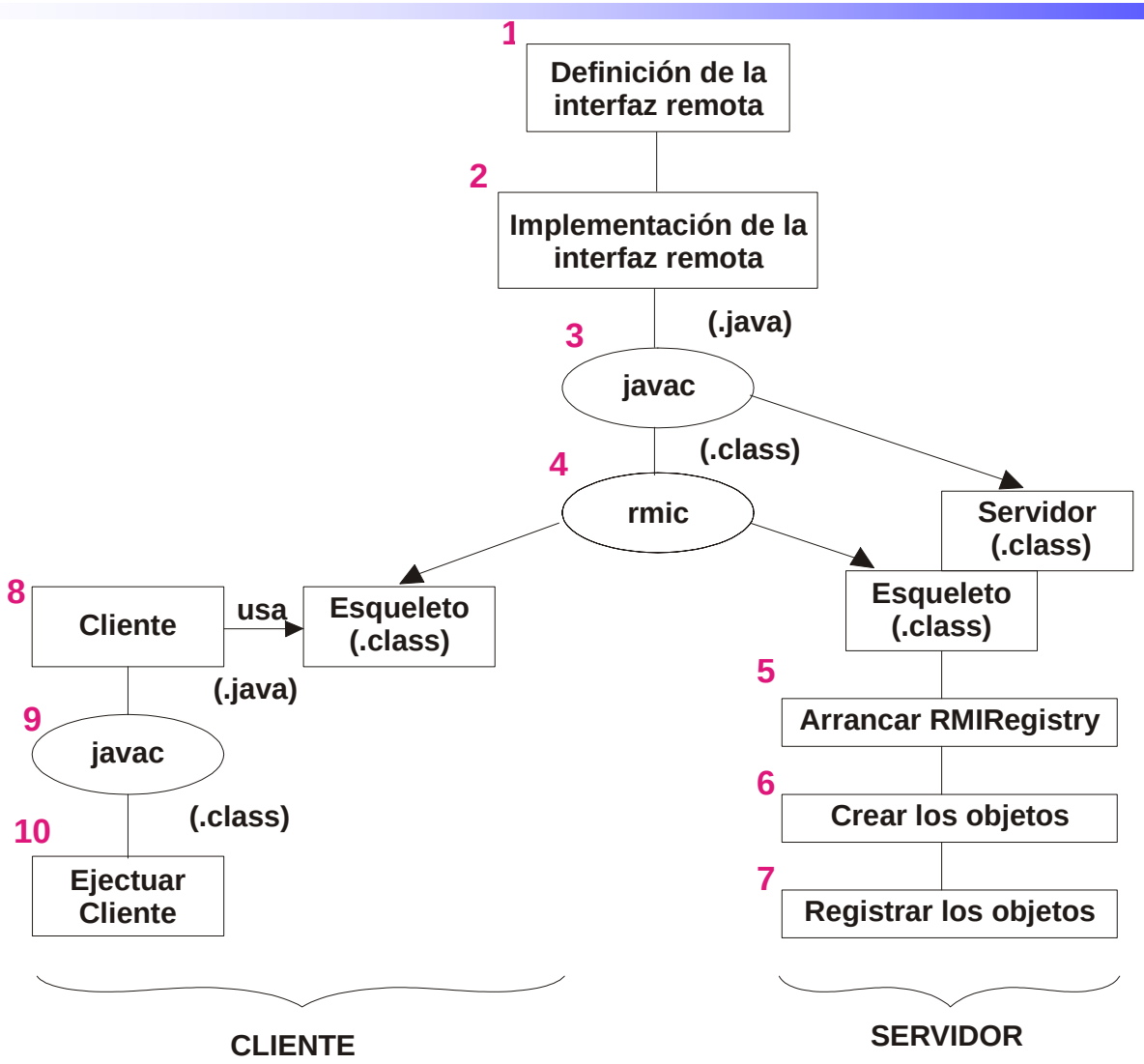
- Servidor de nombres: *java.rmi.Naming*

Ejemplo:

```
Cuenta cnt = new CuentaImpl();
String url = "rmi://java.Sun.COM/cuenta";
// enlazamos una url a un objeto remoto
java.rmi.Naming.bind(url, cnt);
      ....
// búsqueda de la cuenta
cnt=(Cuenta)java.rmi.Naming.lookup(url);
```

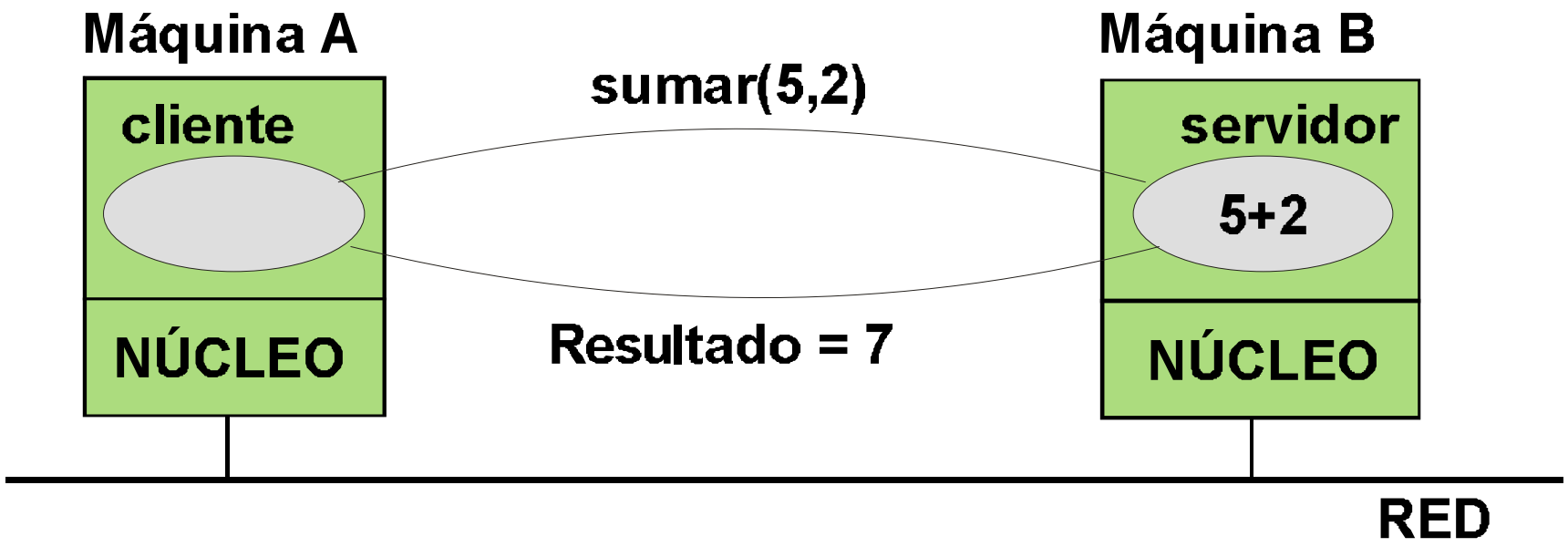

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Desarrollo de Aplicaciones RMI



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Ejemplo


Máquina A
cliente
NÚCLEO
sumar(5,2)
Resultado = 7
Máquina B
servidor
5+2
NÚCLEO
RED

Modelización de la interfaz remota (Sumador)

```
public interface Sumador extends java.rmi.Remote
{

  public int sumar(int a, int b)
       throws java.rmi.RemoteException;

}
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Clase que implementa la interfaz (SumadorImpl)

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class SumadorImpl extends UnicastRemoteObject
   implements Sumador {
    public SumadorImpl(String name) throws RemoteException {
        super();
       try {
        System.out.println("Rebind Object " + name);
            Naming.rebind(name, this);
       } catch (Exception e){
        System.out.println("Exception: " + e.getMessage());
                  e.printStackTrace();
       }
   }
    public int sumar (int a, int b) throws RemoteException {
        return a + b;
    }
}
```

Código del servidor (SumadorServer)

```
import java.rmi.*;
import java.rmi.server.*;

public class SumadorServer  {
   public static void main (String args[]) {
      try {
            SumadorImpl misuma = new
                  SumadorImpl(“rmi://localhost:1099”
                  +”/MiSumador");
            } catch(Exception e) {
            System.err.println("System exception" +
  e);
      }
   }
}
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Registro del servicio

- Antes de arrancar el cliente y el servidor, se debe arrancar el programa *rmiregistry* en el servidor para el servicio de nombres. El puerto que utiliza el *rmiregistry* por defecto es el 1099.
  - `rmiregistry [port_number]`
- El método *rebind* es utilizado normalmente en lugar del método *bind*, porque garantiza que si un objeto remoto se registró previamente con dicho nombre, el nuevo objeto reemplazará al antiguo.
- El método *rebind* almacena en el registro una referencia al objeto con un URL de la forma:
  - `rmi://<nombre máquina>:<número puerto>/<nombre referencia>`

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Código en el cliente (SumadorClient)

```
import java.rmi.registry.*;
import java.rmi.server.*;
import java.rmi.*;
public class SumadorClient {
   public static void main(String args[]){
   int res = 0;
   try {
        System.out.println("Buscando Objeto ");
        Sumador misuma = (Sumador)Naming.lookup("rmi://" +
   args[0] +                                  "/" +"MiSumador");
         res = misuma.sumar(5, 2);
        System.out.println("5 + 2 = " + res);
    }
    catch(Exception e){
              System.err.println(" System exception");    }
    System.exit(0);
    }
}
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Búsqueda

- Cualquier programa que quiera instanciar un objeto remoto debe realizar una búsqueda de la siguiente forma:

```
Sumador misuma = (Sumador)Naming.lookup("rmi://" +  args[0] +
"/"                +"MiSumador");
```

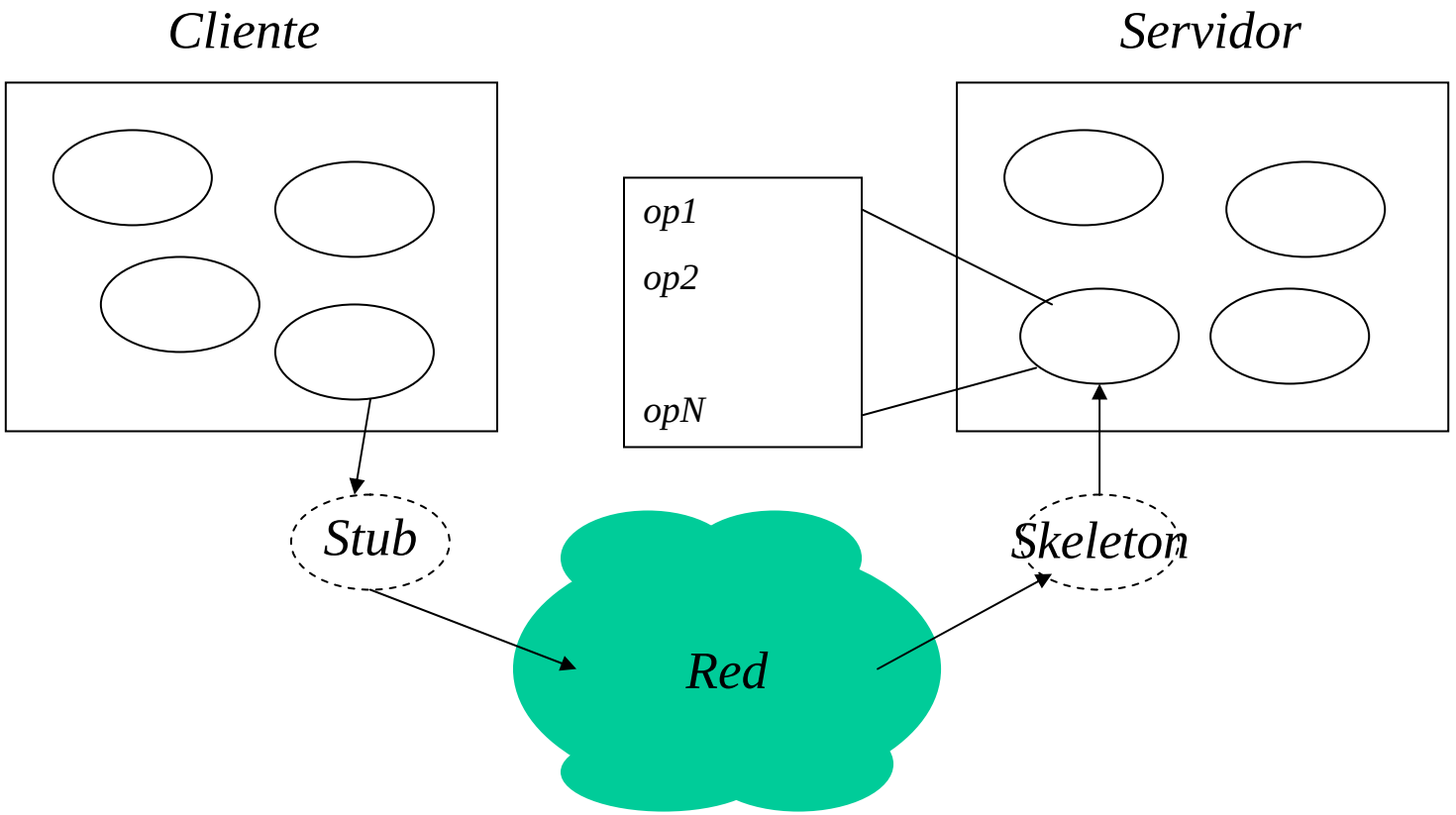
- El método *lookup* devuelve una referencia remota a un objeto que implementa la interfaz remota.
- El método *lookup* interactúa con *rmiregistry*.

Pasos

- Java RMI:
  - Enlace a un nombre: bind(), rebind()
  - Encontrar un objeto y obtener su referencia: lookup()
  - refObj.nombre_met()

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Cuadro general



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

¿Cómo se ejecuta?

- Compilación
  - javac Sumador.java
  - javac SumadorImpl.java
  - javac SumadorClient.java
  - javac SumadorServer.java
- Generación de los esqueletos
  - rmic SumadorImpl
- Ejecución del programa de registro de RMI
  - rmiregistry <número puerto>
  - *Por defecto, en número de puerto es el 1099*
- Ejecución del servidor
  - java SumadorServer
- Ejecución del cliente
  - java SumadorCliente <host-del-servidor>

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Callback de cliente

- Hay escenarios en los que los servidores deben notificar a los clientes la ocurrencia de algún evento. Ejemplo: chat.
  - Problema: llamada a método remoto es unidireccional
- Posibles soluciones:
  - Sondeo (*polling)*: Cada cliente realiza un sondeo al servidor, invocando repetidas veces un método, hasta que éste devuelva un valor *true*.
    - Problema: Técnica muy costosa (recursos del sistema)
  - *Callback*: Cada cliente interesado en la ocurrencia de un evento se registra a sí mismo con el servidor, de modo que el servidor inicia una invocación de un método remoto del cliente cuando ocurra dicho evento.
    - Las invocaciones de los métodos remotos se convierten en bidireccionales

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Extensión de la parte cliente para *callback* de cliente

- El cliente debe proporcionar una interfaz remota: Interfaz remota de cliente

```
public interface CallbackClient extends java.rmi.Remote {
    public String notificame (String mensaje) throws
   java.rmi.RemoteException;
}
```

- Es necesario implementar la interfaz remota de cliente, de forma análoga a la interfaz de servidor (CallbackClientImpl)
- En la clase cliente se debe añadir código para que instancie un objeto de la implementación de la interfaz remota de cliente y que se registre para callback (método implementado por el servidor):

```
CallbackServer cs = (CallbackServer) Naming.lookup(URLregistro);
CallbackClient objCallback = new CallbackClientImpl();
cs.registrarCallback(objCallback);
```

Extensión de la parte servidora para *callback* de cliente

- Añadir el método remoto para que el cliente se registre para *callback*

```
public void registrarCallback (CallbackClient
objCallbackClient) throws java.rmi.RemoteException;
```

- Se puede proporcionar un método `eliminarRegistroCallback` para poder cancelar el registro
- Ambos métodos modifican una estructura común (por ejemplo, un objeto Vector) que contiene referencias a los *callbacks* de clientes. Se utilizan métodos *synchronized* para acceder a la estructura en exclusión mutua.

Descarga dinámica de resguardo

- Mecanismo que permite que los clientes obtengan dinámicamente los resguardos necesarios
  - Elimina la necesidad de copia de la clase del resguardo en la máquina cliente
  - Se transmite bajo demanda desde un servidor web a la máquina cliente (Similar a la descarga de los applets)
- El servidor exporta un objeto a través del registro RMI (registro de una referencia remota al objeto mediante nombre simbólico) e indica el URL donde se almacena la clase resguardo.
- La clase resguardo descargada no es persistente
  - Se libera cuando la sesión del cliente finaliza

Comparación RMI y Sockets

- Los sockets están más cercanos al sistema operativo, lo que implica una menor sobrecarga de ejecución.
- RMI proporciona mayor abstracción, lo que facilita el desarrollo de software. RMI es un buen candidato para el desarrollo de prototipos.
- Los sockets suelen ser independientes de plataforma y lenguaje.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Sistemas Operativos Distribuidos

Entornos de Objetos Distribuidos

- CORBA

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

OMG

(Object Management Group)

Conjunto de organizaciones que cooperan en la definición de estándares para la interoperabilidad en entornos heteregéneos.

Fundado en 1989, en la actualidad lo componen más de 700 empresas y otros organismos.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

OMA

(Object Management Architecture)

Arquitectura de referencia sobre cual se pueden definir aplicaciones distribuidas sobre un entorno heteregéneo.

CORBA es la tecnología asociada a esta arquitectura genérica.

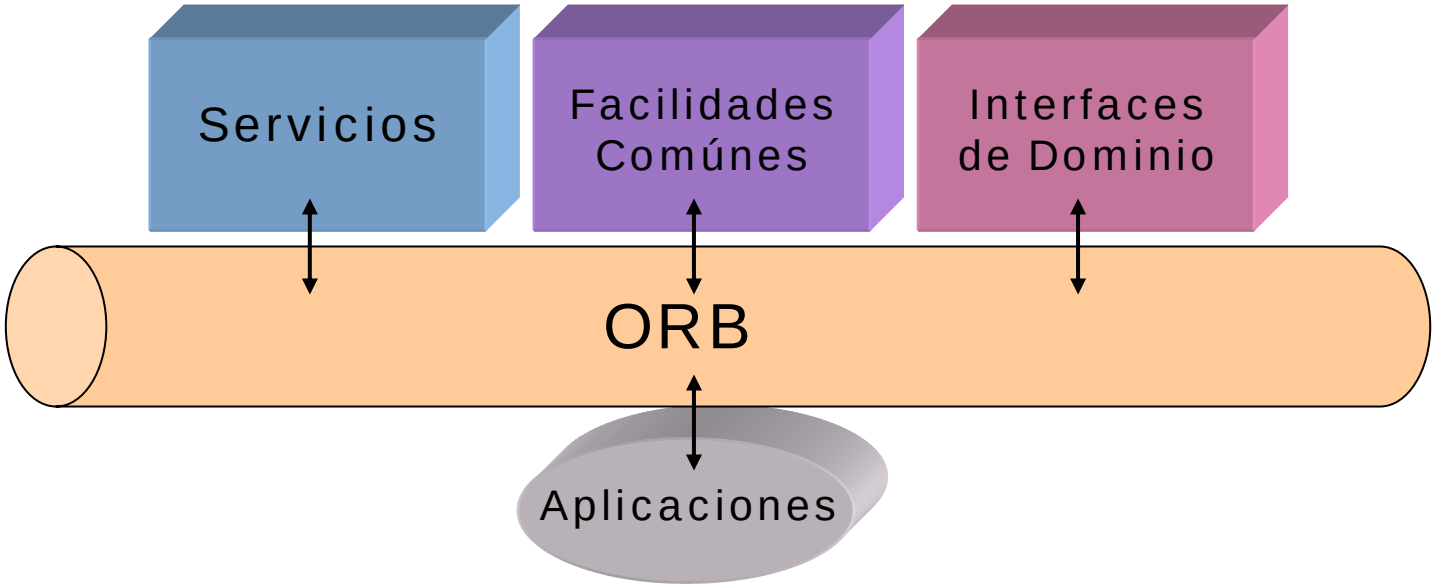
Formalmente esta dividida en una serie de modelos:

- Modelo de Objetos
- Modelo de Interacción
- ...

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

OMA

Una aplicación definida sobre OMA está compuesta por una serie de objetos distribuidos que cooperan entre sí. Estos objetos se clasifican en los siguientes grupos:



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

OMA

Servicios:

- Proporcionan funciones elementales necesarias para cualquier tipo de entorno distribuido, independientemente del entorno de aplicación.
- Los tipos de funciones proporcionados son cuestiones tales como la resolución de nombres, la notificación asíncrona de eventos o la creación y migración de objetos.
- Concede un valor añadido sobre otras tecnologías (por ejemplo RMI).
- Están pensados para grandes sistemas.

OMA

Aplicaciones:

- El resto de funciones requeridas por una aplicación en concreto. Es el único grupo de objetos que OMG no define, pues está compuesto por los objetos propios de cada caso concreto.

- Estos son los objetos que un sistema concreto tiene que desarrollar. El resto (servicios, facilidades) pueden venir dentro del entorno de desarrollo.

OMA

ORB:

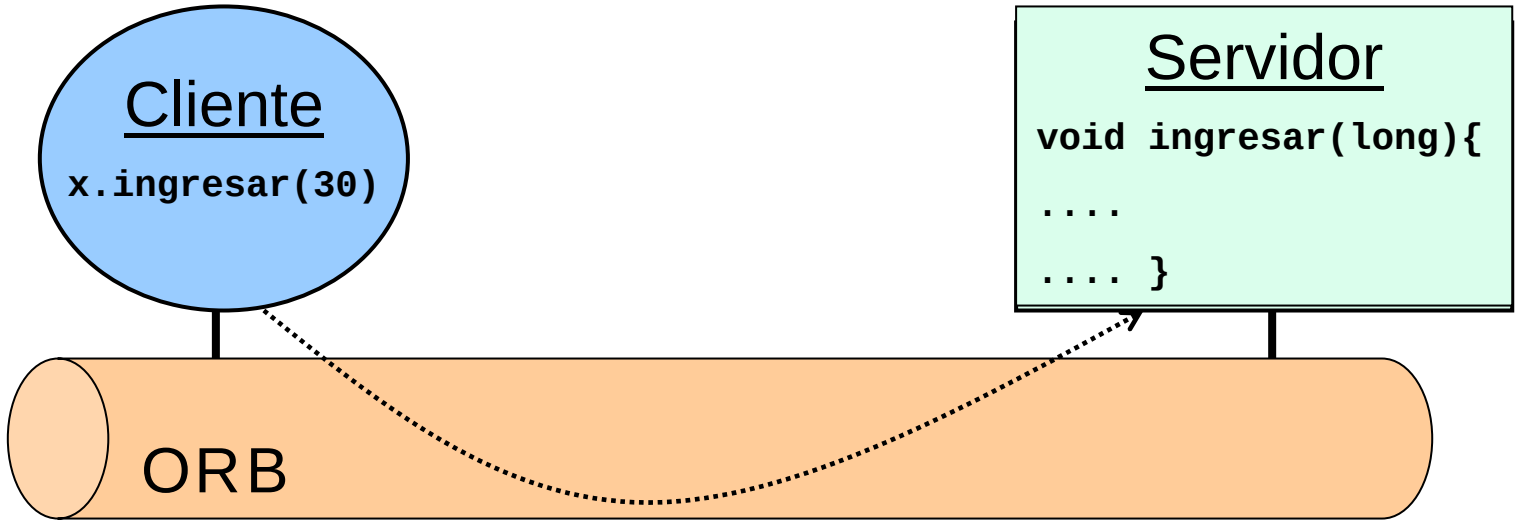
- (*Object Request Broker*)

- Es el elemento central de la arquitectura. Proporciona las funcionalidades de interconexión entre los objetos distribuidos (servicios, facilidades y objetos de aplicación) que forman una aplicación.

- Representa un bus de comunicación entre objetos.

ORB

Para posibilitar la comunicación entre dos objetos cualesquiera de una aplicación se realiza por medio del ORB. El escenario de aplicación elemental es:



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

IDL de CORBA

(*Interface Definition Language*)

Es el lenguaje mediante el cual se describen los métodos que un determinado objeto del entorno proporciona al resto de elementos del mismo.

```
interface Cuenta
{
  void ingresar(in long cantidad);
  void retirar(in long cantidad);
  long balance();
};
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

IDL de CORBA

Language Mappings:

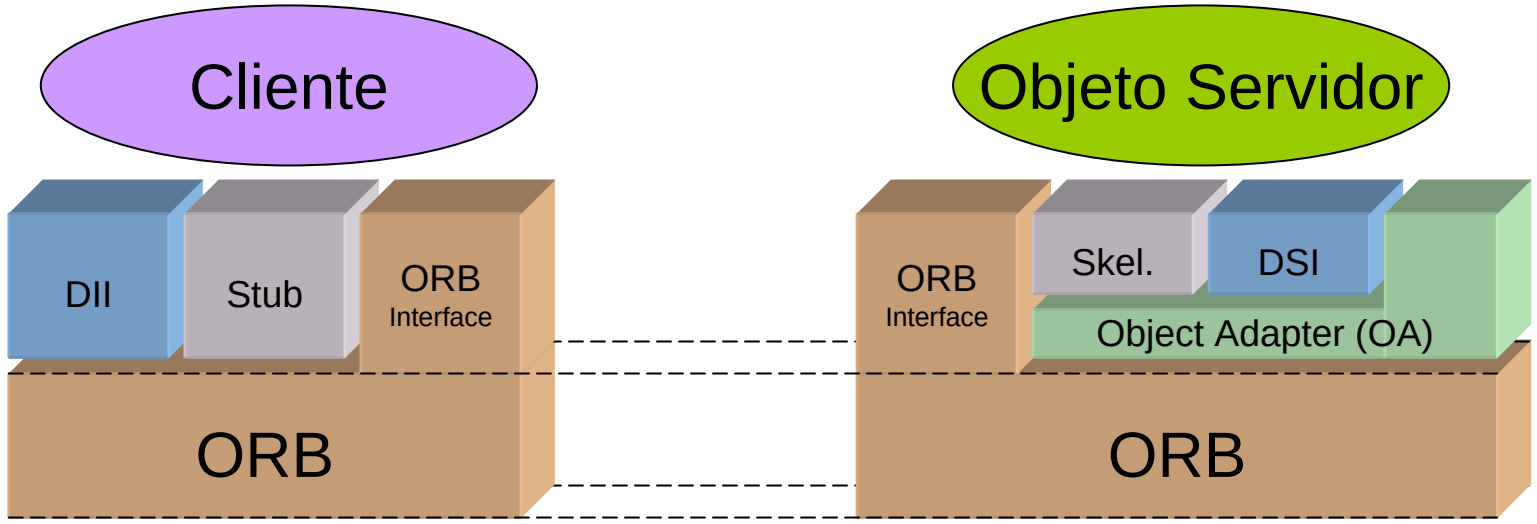
- Traducen la definición IDL a un lenguaje de programación como:
  - C++,
  - Ada,
  - COBOL,
  - SmallTalk,
  - Java,
  - ...

- Este proceso genera dos fragmentos de código denominados *Stub* y *Skeleton* que representan el código de cliente y servidor respectivamente.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Componentes de un ORB

La arquitectura completa de comunicaciones de CORBA es la siguiente:



Componentes de un ORB

Stub:

- Código cliente asociado al objeto remoto con el que se desea interactuar. Simula para el cliente los métodos del objeto remoto, asociando a cada uno de los métodos una serie de funciones que realizan la comunicación con el objeto servidor.

Skeleton:

- Código servidor asociado al objeto. Representa el elemento análogo al stub del cliente. Se encarga de simular la petición remota del cliente como una petición local a la implementación real del objeto.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Componentes de un ORB

DII:

- (*Dynamic Invocation Interface*)
- Alternativa al uso de stubs estáticos que permite que un cliente solicite peticiones a servidores cuyos interfaces se desconocían en fase de compilación.

DSI:

- (*Dynamic Skeleton Interface*)
- Alternativa dinámica al uso de skeletons estáticos definidos en tiempo de compilación del objeto. Es usado por servidores que durante su ejecución pueden arrancar diferentes objetos que pueden ser desconocidos cuando se compiló el servidor.

Componentes de un ORB

ORB/Interface ORB:

- Elemento encargado de (entre otras) las tareas asociadas a la interconexión entre la computadora cliente y servidor, de forma independiente de las arquitecturas hardware y SSOO.
- Debido a que tanto clientes como servidores pueden requerir de ciertas funcionalidades del ORB, ambos son capaces de acceder a las mismas por medio de una interfaz.

Las dos principales responsabilidades del ORB son:

- Localización de objetos: El cliente desconoce la computadora donde se encuentra el objeto remoto.
- Comunicación entre cliente y servidor: De forma independiente de protocolos de comunicación o características de implementación (lenguaje, sistema operativo, ...)

Componentes de un ORB

Adaptador de Objetos:

- En este elemento se registran todos los objetos que sirven en un determinado nodo. Es el encargado de mantener todas las referencias de los objetos que sirven en una determinada computadora de forma que cuando llega una petición a un método es capaz de redirigirla al código del skeleton adecuado.

Existen dos tipos de Adaptadores de Objetos especificados por OMG:

- BOA: (Basic Object Adapter).
- POA: (Portable Object Adapter).

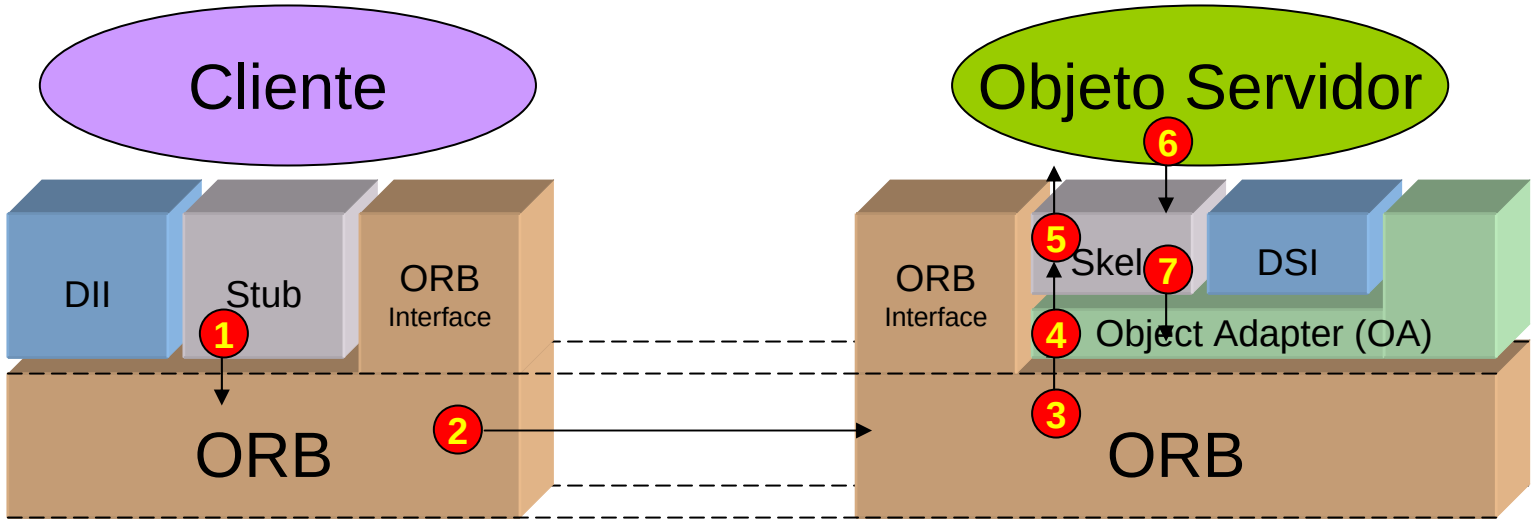
Comunicación vía CORBA

Pasos de una comunicación:

1- El cliente invoca el método asociado en el stub que realiza el proceso de marshalling. (Stub cliente)

2- El stub solicita del ORB la transmisión de la petición hacia el objeto servidor. (ORB cliente)

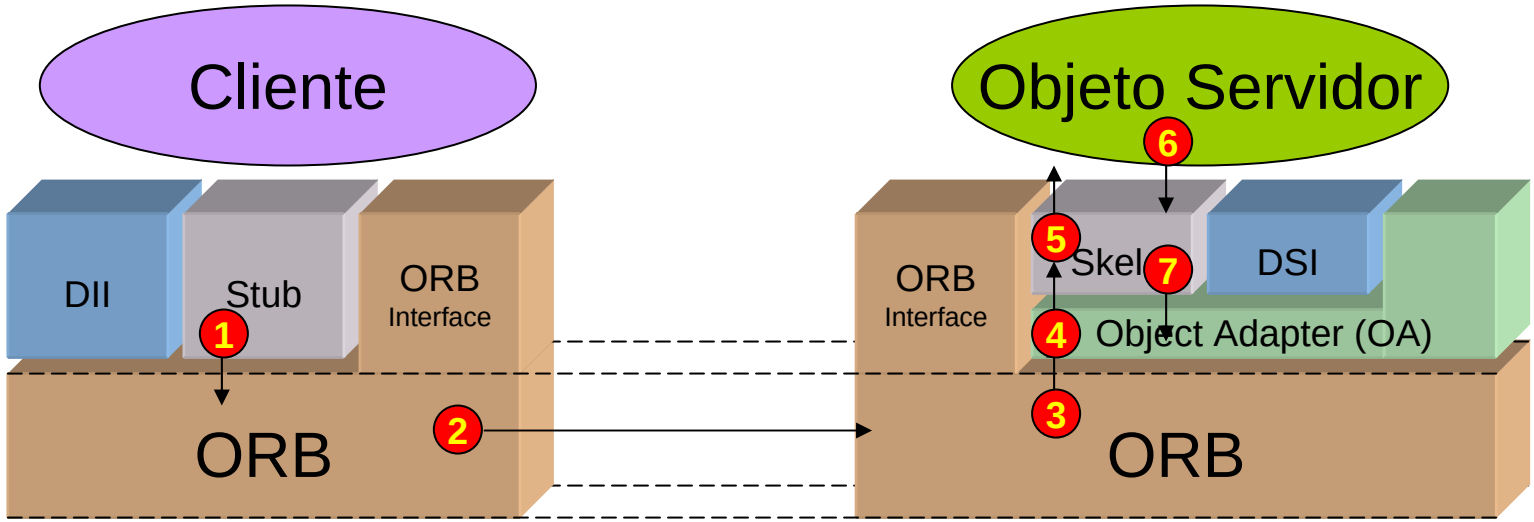
3- El ORB del servidor toma la petición y la transmite el Adaptador de Objetos asociado, por lo general sólo hay uno. (ORB servidor)



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Comunicación vía CORBA

4- El Adaptador de Objetos resuelve cuál es el objeto invocado, y dentro de dicho objeto cuál es el método solicitado (Adaptador de Objetos)
5- El skeleton del servidor realiza el proceso de de-marshalling de los argumentos e invoca a la implementación del objeto. (Skeleton servidor)
6- La implementación del objeto se ejecuta y los resultados y/o parámetros de salida se retornan al skeleton. (Implementación del objeto)
7- El proceso de retorno de los resultados es análogo.



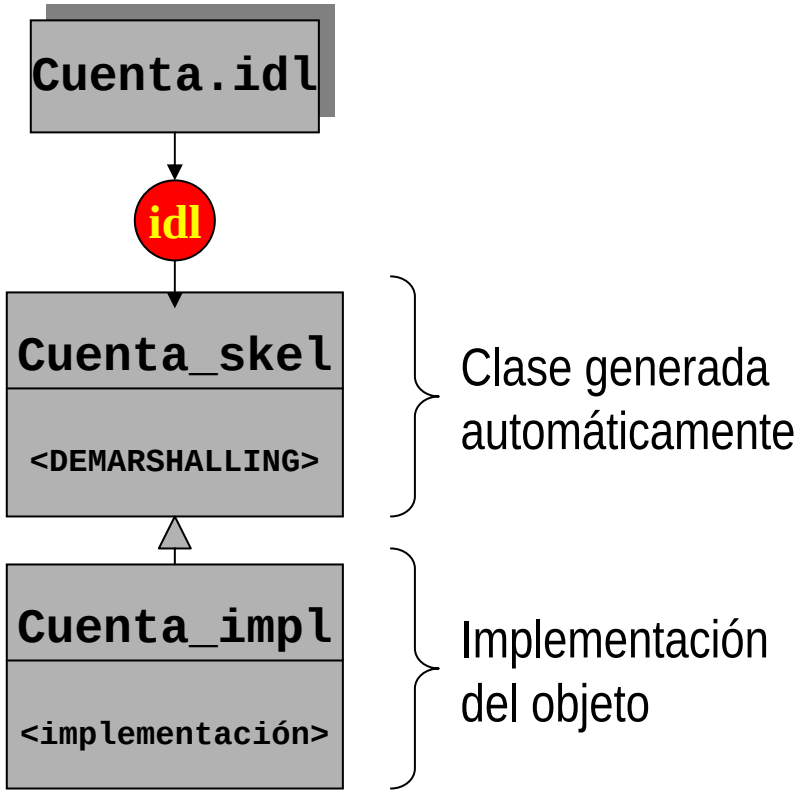
Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Localización de Objetos

- Los objetos de servicio de una aplicación CORBA se encuentran identificados por medio de una referencia única (Identificador de Objeto).
- Esta referencia es generada al activar un objeto en el Adaptador de Objetos.
- Por medio de esta referencia el ORB es capaz de localizar la computadora y el Adaptador de Objetos donde se encuentra, y éste último es capaz de identificar el objeto concreto dentro del Adaptador.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Implementación del Servidor



La implementación del objeto se diseña como una subclase de la clase generada por el compilador (el *skeleton*) de IDL en base a la definición.

Si el lenguaje usado para la implementación no soporta objetos el mecanismo es diferente.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Servicios CORBA

Conjunto de objetos o grupos de objetos, que proporcionan una serie de funcionalidades elementales. Estos objetos están definidos de forma estándar (interfaces IDL concretas).

- Sus especificaciones se encuentran recogidas en los documentos COSS (Common Object Services Specifications).
- Los servicios definidos son:

- Servicio de Nombres
- Servicio de Eventos
- Servicio de Ciclo de Vida
- Servicio de Objetos Persistentes
- Servicio de Transacciones
- Servicio de Control de Concurrencia
- Servicio de Relación
- Servicio de Externalización
- Servicio de Consulta
- Servicio de Licencias
- Servicio de Propiedad
- Servicio de Tiempo
- Servicio de Seguridad
- Servicio de Negociación
- Servicio de Colección de Objetos

Uso del Servicio de Nombres

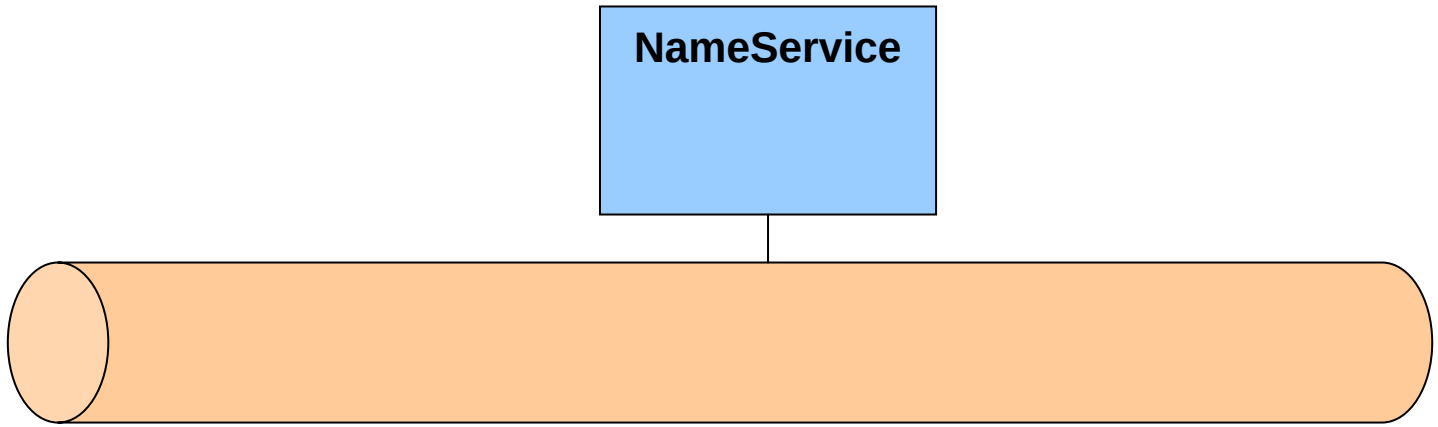
Permite asociar un nombre a una referencia de objeto. De esta forma los objetos al activarse pueden darse de alta en el servidor, permitiendo que otros objetos los obtengan su referencia en base a dicho nombre.

Los nombres de los objetos se encuentran organizados en una jerarquía de *contextos de nombre*.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

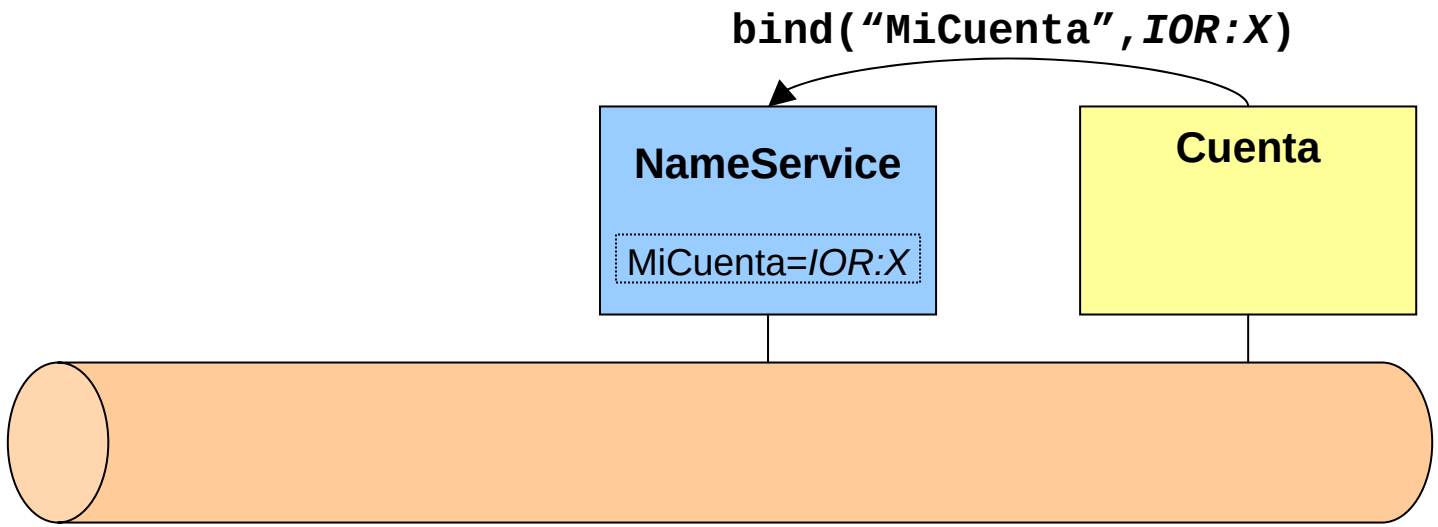
Servicio de Nombres

El Servidos de Nombres, al igual que todo objeto del sistema se encuentra previamente activo.



Servicio de Nombres

Un nuevo objeto se arranca y se registra en el servidor de nombres:

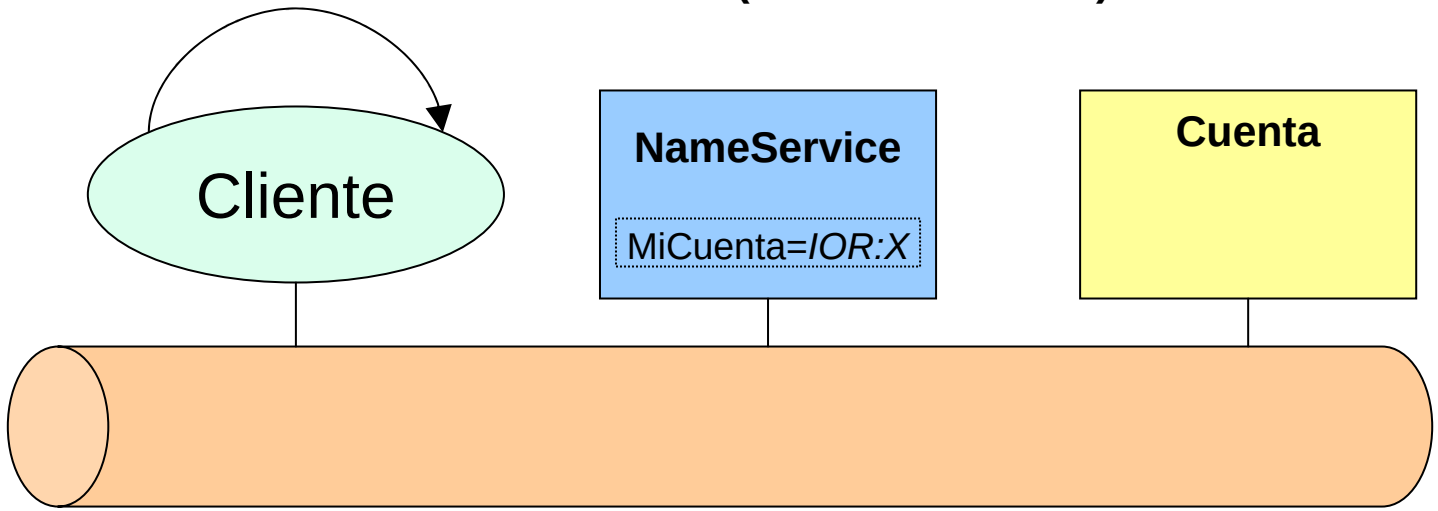


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Servicio de Nombres

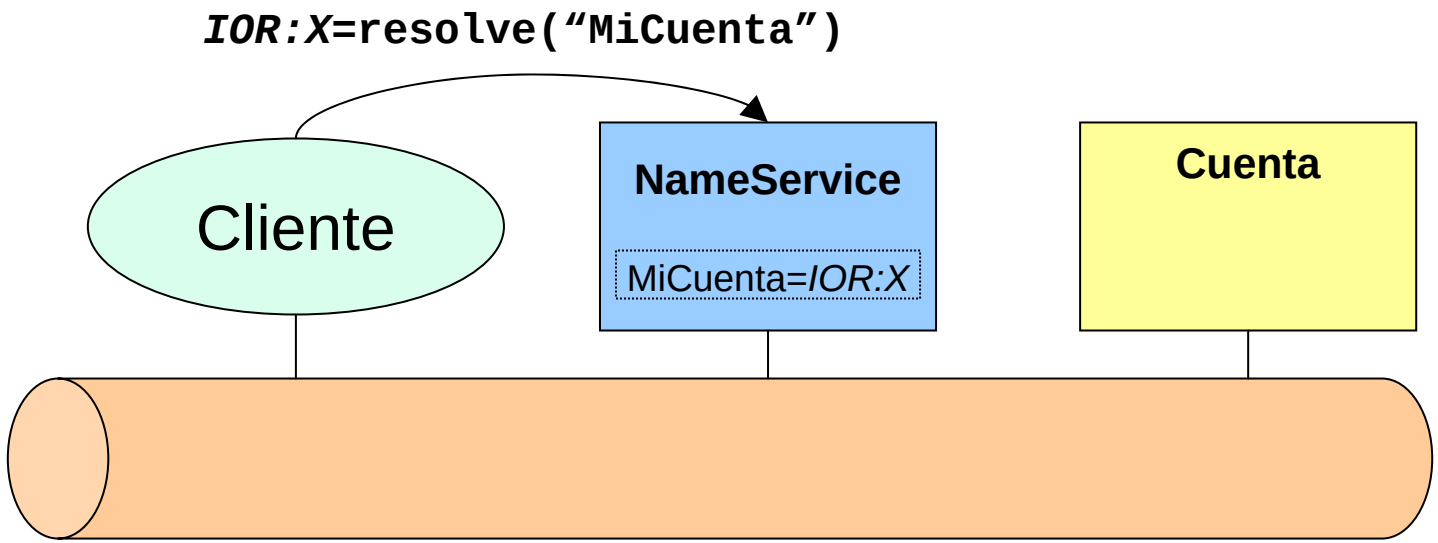
Un cliente localiza al servidor de nombres. Suele existir una función interna del ORB para localizar al servidor de nombres (**`resolve_initial_references`**) .

***`IOR:NS`*`=resolve_initial_references("NameService")`**



Servicio de Nombres

El cliente le pide al servidor de nombres que resuelva el nombre. Así obtiene la referencia al objeto buscado.



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Servicio de Negociación

Este servicio también permite obtener referencias a objetos usando otra información:

- Los objetos se exportan en el servidor con una serie de características del servicio que proporcionan.
- Los clientes consultan con el servidor cuáles son los objetos ofertados con una serie de características.

Un cliente que buscase un servicio de impresión podría construir una consulta del tipo:

```
Service=Printer AND
PrinterType=HP AND
OS!=WinNT
```

A lo cual el servidor le indicará los objetos que existen con dichas características.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Comparativa

CORBA vs DCOM

- CORBA es un estándar abierto y no propietario.
- CORBA proporciona soporte para diversos SO.
- CORBA es más completo y flexible.
- CORBA da una salida a los *legacy systems*

- DCOM esta integrado con la tecnología *MicroSoft*.
- DCOM ha tenido una fuerte penetración en el mercado.

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Comparativa

CORBA vs RMI

- CORBA permite una mayor heterogeneidad en el desarrollo de aplicaciones (RMI sólo se puede desarrollar con Java).
- CORBA ademas de las funcionalidades de comunicación, proporciona servicios.
- RMI funciona sobre CORBA (IIOP).

- RMI es mucho más sencillo y cómodo de usar.
- RMI permite un desarrollo rápido de prototipos.

Sistemas Operativos Distribuidos

Introducción a los Servicios Web

(*Web Services*)

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Evolución de la Web

- Pasado: Web de documentos
  - Páginas estáticas
  - Web como un enorme repositorio de información
  - Tecnologías: HTTP + HTML
- Presente: Web de aplicaciones
  - Páginas dinámicamente generadas por aplicaciones web
  - Aplicaciones exportan su interfaz a los usuarios a través de la Web
  - Entorno de transacciones comerciales (*Business to consumer*, B2C)
  - Tecnologías: CGI, ASP, PHP, JSP, *servlets*, ...
- Futuro (ya está aquí): Web de servicios (funciones/métodos)
  - "Bibliotecas" ofrecen servicios a **programas** (no a usuarios)
  - Web como una enorme API de servicios (Web de componentes)
  - Empresas de valor añadido (*Business to business*, B2B)
  - Base de Sistemas distribuidos sobre Internet
    - Servicio web: RPC sobre la Web usando XML

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Aplicaciones web: Escenario típico

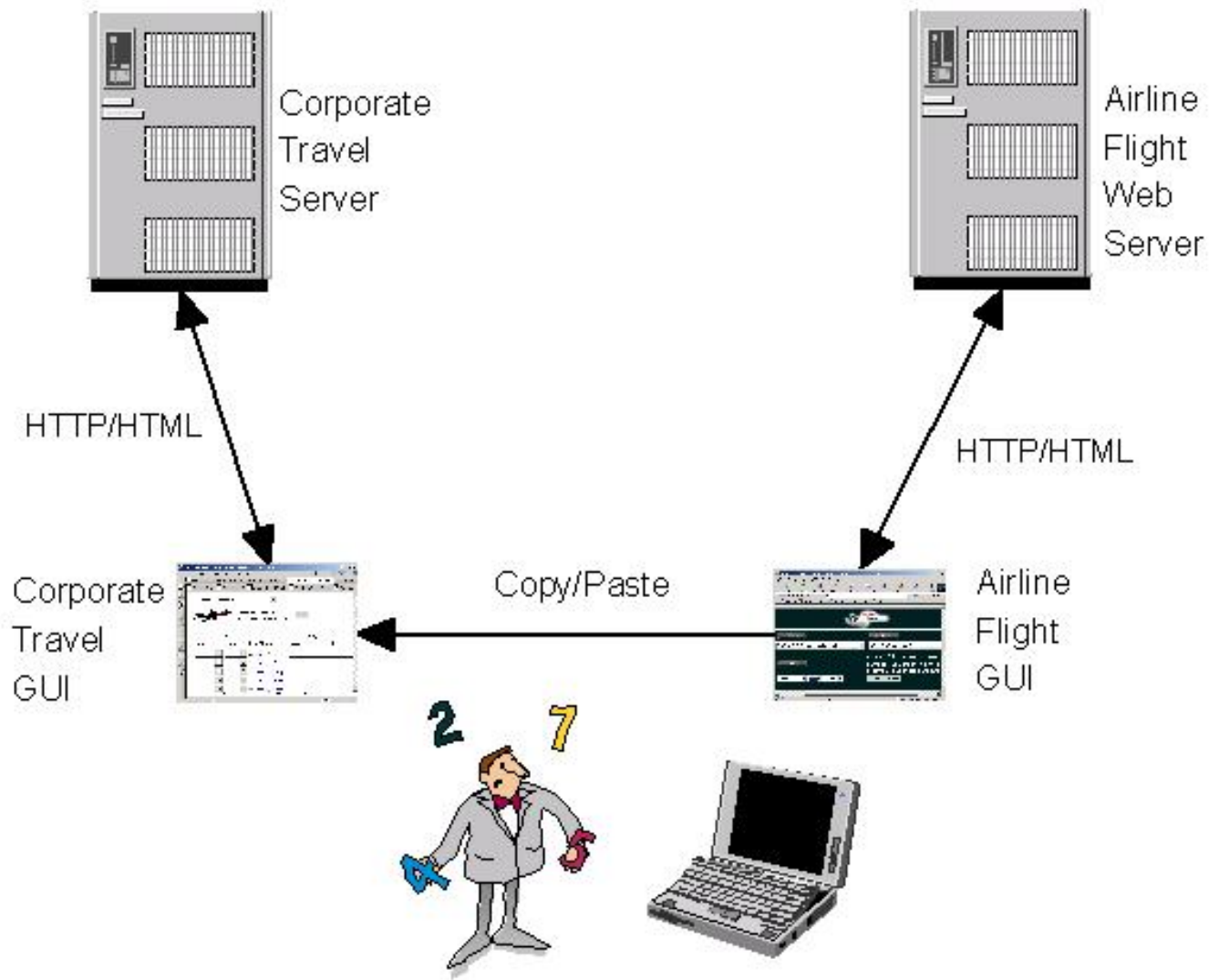


Figura extraída de "*Understanding Web Services*": http://www7.software.ibm.com/vad.nsf/Data/Document4362

Servicios web: Escenario típico

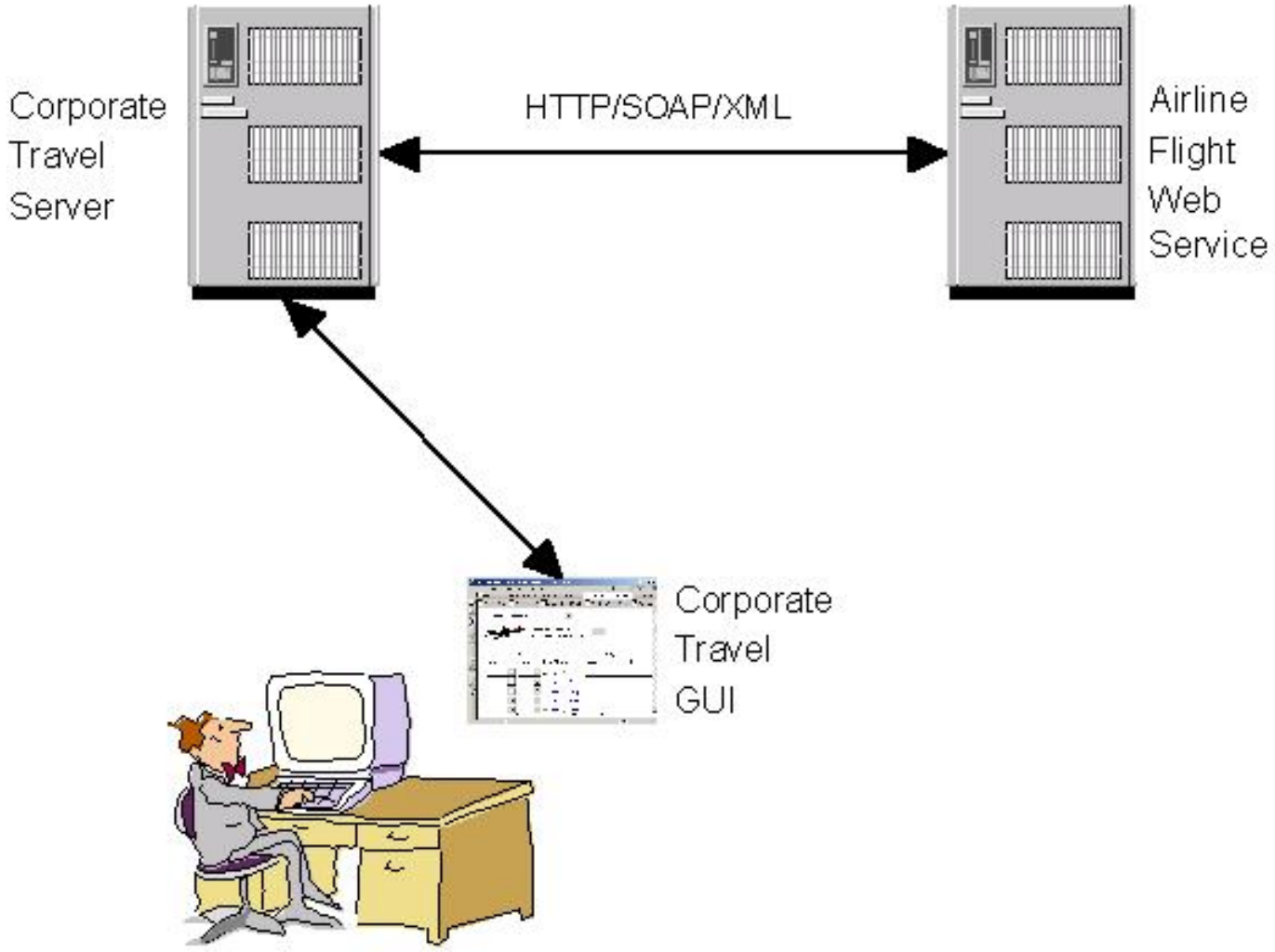


Figura extraída de "*Understanding Web Services*":
http://www7.software.ibm.com/vad.nsf/Data/Document4362

Integración de servicios web

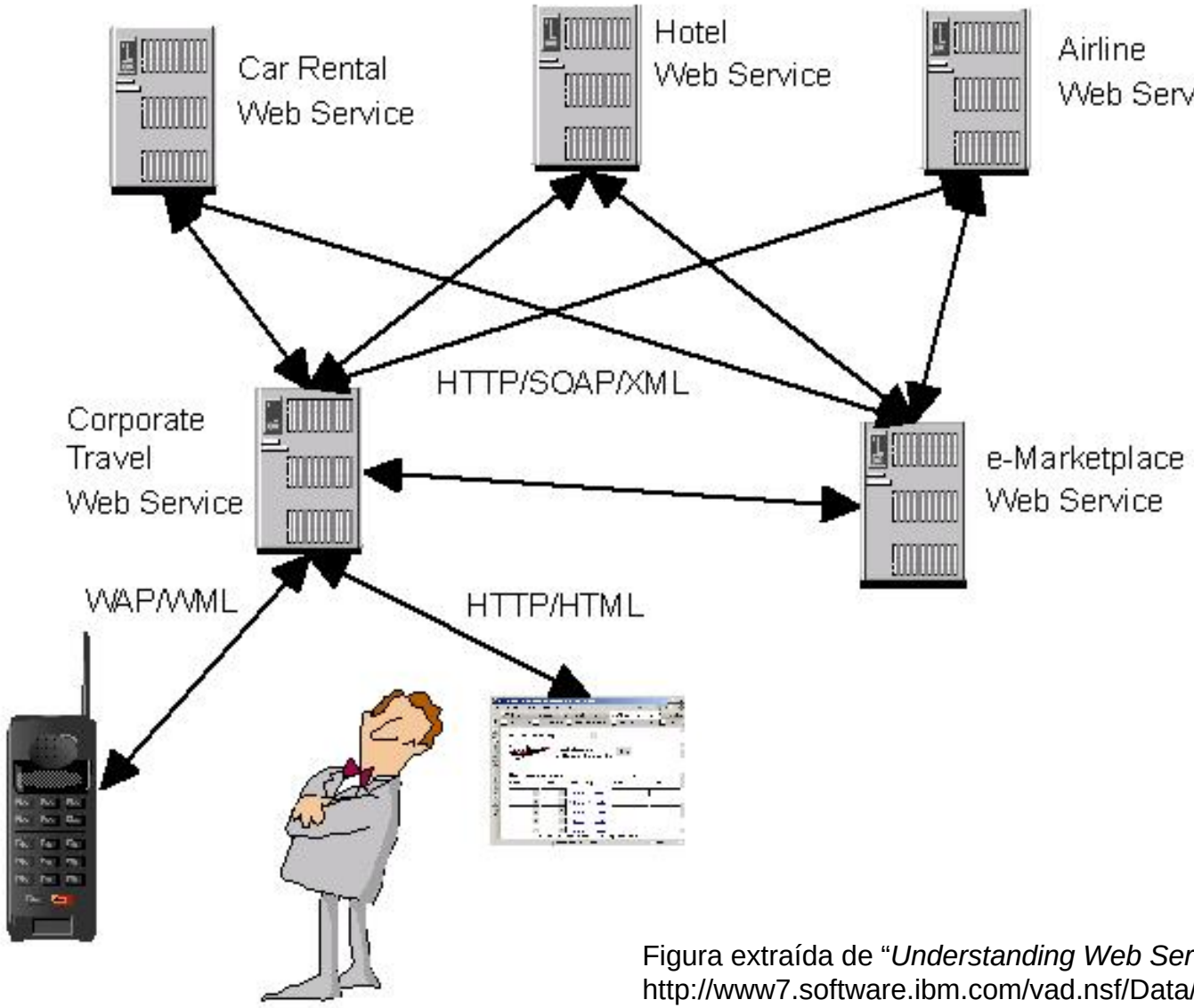


Figura extraída de "*Understanding Web Services*":
http://www7.software.ibm.com/vad.nsf/Data/Document4362

Definición de servicio web

- Módulo que exporta un conjunto de funciones (métodos) a aplicaciones a través de la Web proporcionando independencia de plataformas hardware/software
- Similar a RPC o RMI pero integrado en la Web
  - ¿Reinventando la rueda? → ¿Por qué no usar CORBA?
- Estandarización controlada por un grupo del W3C:
  - http://www.w3.org/2002/ws/
- Mismas cuestiones que con RPC/RMI:
  - ¿Qué protocolo de transporte se usa? → HTTP
  - ¿Qué formato de representación se utiliza? → XML
  - ¿Qué protocolo de comunicación se usa? → SOAP
  - ¿Cómo se especifican los servicios exportados (IDL)? → WSDL
  - ¿Cómo localiza el cliente al servidor (*binding*)? → UDDI

Protocolo de transporte: HTTP

- Uso típico de operación POST de HTTP:
  - datos de formulario y página de respuesta

```
POST /~ssoo/consultaBD.cgi HTTP/1.0
Content-length: 76
.....

DNI=87654321&MAT=980000&Asignatura=sod&Curso=2002&Convocatoria=Jun&Tipo=acta
```

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=iso-8859-1
.....

<HTML>
```

- Uso de POST para petición y respuesta de RPC
  - Universalmente disponible
  - Atraviesa el *firewall* de la organización

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Formato de representación: XML

- Información de RPC codificada en XML
  - Muy flexible y potente
  - XML Schema permite definir con precisión los tipos de datos
- Ej: *float GetLastTradePrice(string symbol);*

Petición:

```
<GetLastTradePrice>
      <symbol>DIS</symbol>
</GetLastTradePrice>
```

Respuesta:

```
<GetLastTradePriceResponse>
      <Price>34.5</Price>
</GetLastTradePriceResponse>
```

Esquema:

```
<element name="GetLastTradePrice">
<complexType><all>
      <element name="symbol" type="string"/>
</all></complexType>
</element>
<element name="GetLastTradePriceResponse">
<complexType><all>
      <element name="Price" type="float"/>
</all></complexType>
</element>
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Protocolo de comunicación: SOAP

- *Simple Object Access Protocol* (*Candidate Recommendation*)
- SOAP = HTTP + XML
  - Especifica cómo mandar mensajes XML sobre HTTP
  - Define el contenedor del mensaje (tambien en XML)
  - Protocolo general, no sólo para RPC, también unidireccional
- Estructura de mensaje contenedor SOAP:
  - Sobre (*Envelope*): Cabecera (*Header*) [opcional] + Cuerpo (*Body*)
    - Cabecera : info. complementaria (p.ej. en RPC un ID de transacción)
    - Cuerpo: contiene el mensaje original
- SOAP para RPC:
  - En petición: Identificador en POST identifica destino de RPC
- Seguridad:
  - Usando HTTPS (SSL)
  - Nueva propuesta: *WS-Security*

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Un ejemplo de SOAP en RPC

Petición

```
POST /StockQuote HTTP/1.1
.....
<SOAP-ENV:Envelope   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
      <m:GetLastTradePrice xmlns:m="http://example.com/stockquote.xsd">
            <symbol>DIS</symbol>
      </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Respuesta

```
HTTP/1.1 200 OK
.....
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="http://example.com/stockquote.xsd">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Definición de interfaz de servicio: WSDL

- *Web Service Description Language* (*Working Draft*)
- IDL para servicios Web basado en XML
- Documento WSDL describe servicio web:
  - Tipos de datos (XML Schema)
  - Funciones exportadas y sus mensajes de petición y respuesta
  - Protocolos usados: típicamente SOAP sobre HTTP
  - Dirección de servicio → URL con servidor y "componente"
    - P. ej. *http://www.stockquoteserver.com/StockQuote*
- Normalmente, generado automáticamente a partir de código de servicios

Desarrollo de un servicio Web

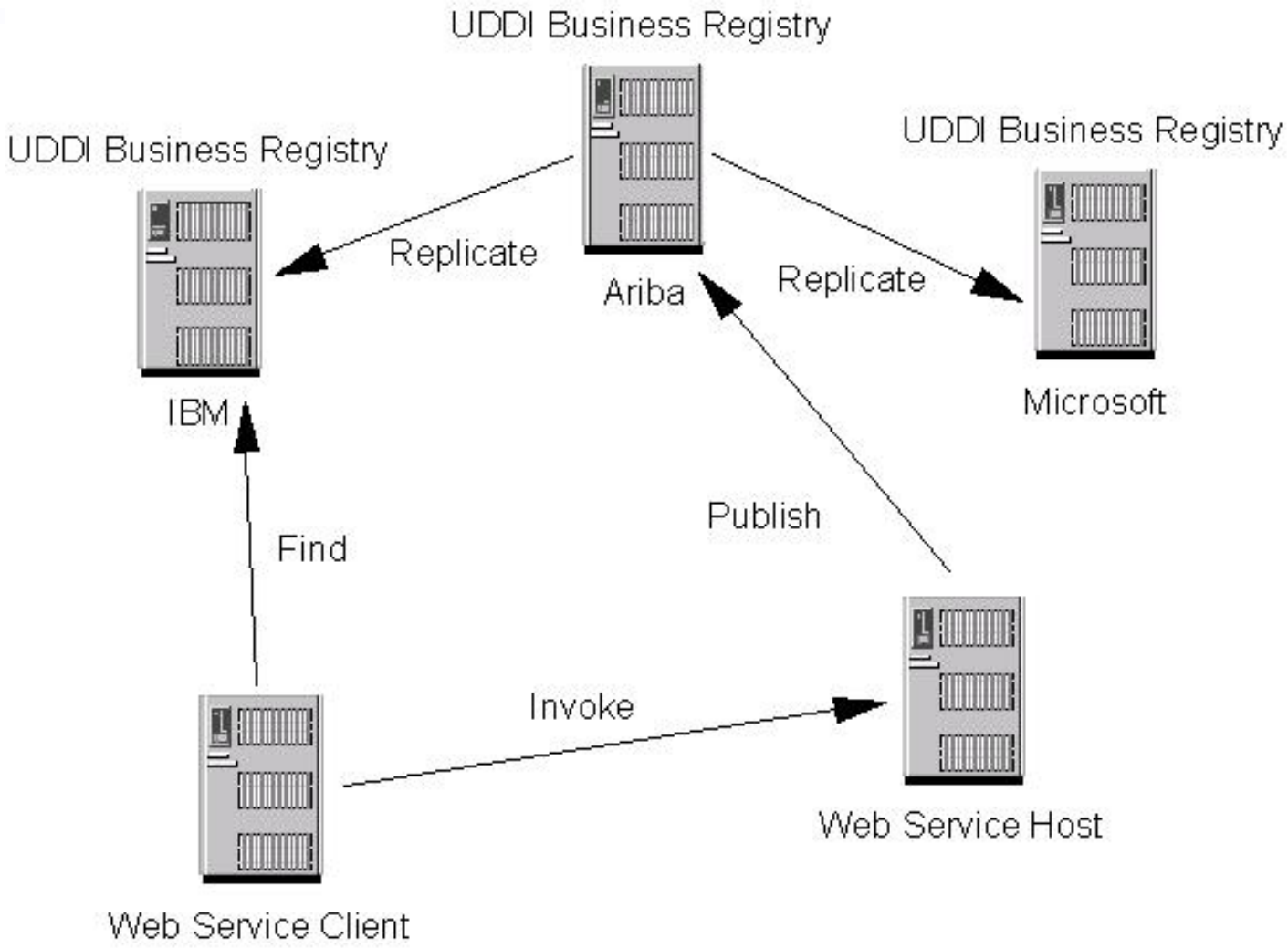
- Programación de biblioteca de servicio
  - En algunos entornos hay que incluir información específica
    - En VisualStudio .Net: etiqueta *[WebMethod]* sobre métodos exportados
- Generación automática de fichero WSDL
  - Generalmente, dentro de la generación de aplicación de servicio
    - En VisualStudio .Net: Proyecto de tipo *Web Service*
- En servidor: fichero WSDL informa sobre cómo activar servicio
  - Normalmente, lo hace un servidor web con soporte de servicios web
- Desarrollo de cliente:
  - Obtener fichero WSDL y generar proxy para aplicación cliente
    - En VisualStudio .Net: "*Add Web Reference*"

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

Localización del servicio: UDDI

- *Universal Description, Discovery, and Integration*
  - No estándar: Propuesta inicial de Microsoft, IBM y Ariba
- Registro distribuido de servicios web ofrecidos por empresas
- Información clasificada en 3 categorías (guías):
  - Páginas blancas: Datos de la empresa
  - Páginas amarillas: Clasificación por tipo de actividas
  - Páginas verdes: Descripción de servicios web (WSDL)
- Se accede a su vez como un servicio web
- Puede consultarse en tiempo de desarrollo o incluso dinámicamente en tiempo de ejecución
- Permite búsquedas por distintos criterios
  - Tipo de actividad, tipo de servicio, localización geográfica

Registro de un servicio web



Extesiones de protocolos

- ASAP (Asynchronous Service Access Protocol):
  - Solicitudes asíncronas (envío cliente -> servidor).
  - Extensión de SOAP.
  - Pensadas para transacciones de larga duración.

- DIME (Direct Internet Message Encapsulation):
  - Optimización seleccionando la codificación de porciones del mensaje.
  - Extensión de SOAP / SOAP MTOM.
  - Empaquetado más ligero.

Servicios web vs. RPC/RMI

- Servicio Web similar a RPC/RMI (Corba, DCOM)
  - ¿Hay un “ganador”? ¿Reinventando la rueda?
- Convivencia: Distintos ámbitos de aplicación
- Servicios web
  - Entornos de interacción “débilmente acoplados”
    - Exportar servicios fuera de la organización
    - Integrar aplicaciones de la empresa
  - Más estándar y extendido, menos problemas con *firewalls*
- RPC/RMI “tradicionales”
  - Aplicación distribuida “fuertemente acoplada”
  - Más funcionalidad y eficiencia
- Ejemplo de escenario de convivencia:
  - Exportar un servicio interno CORBA mediante un servicio web

Entornos de desarrollo de servicios web

- Número creciente de entornos de desarrollo
- Algunas implementaciones de interés:
  - .Net de Microsoft
  - *Web Services Project* de Apache
  - *Java Web Services Developer Pack*
  - *IBM WebSphere SDK for Web services* (WSDK)
  - WASP de Systinet
- Cursos sobre SOAP, WSDL y otras tecnologías web:
  - http://www.w3schools.com/