Arquitecturas Orientadas a Servicios

INTRODUCCIÓN

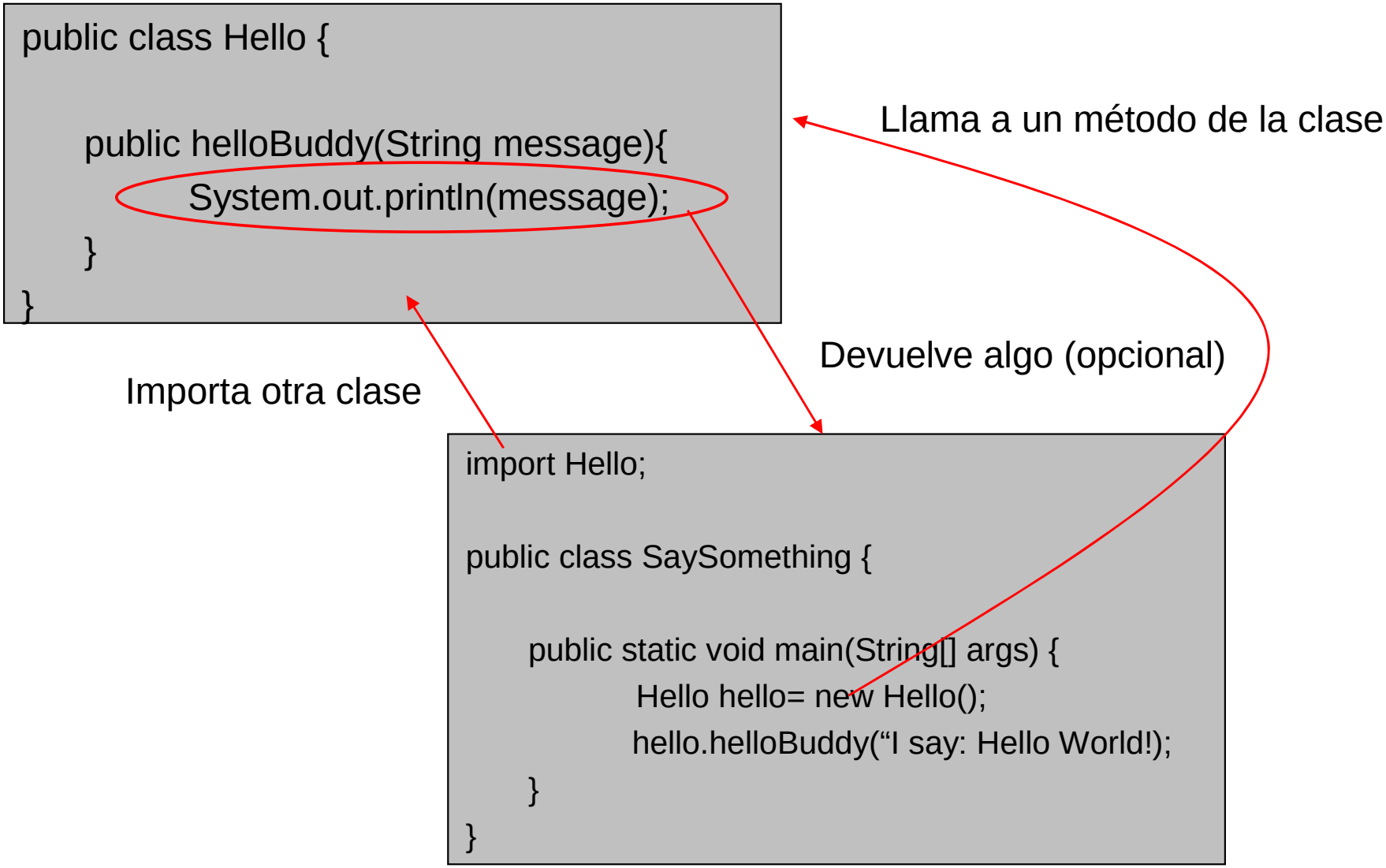
Manuel Lama Penín

manuel.lama@usc.es

Grupo de Sistemas Inteligentes
Departamento de Electrónica e Computación
Universidade de Santiago de Compostela

Descripción del problema

Un método invoca la ejecución de otros métodos para **obtener algo** que no puede alcanzar por sí mismo.



Descripción del problema

Una clase **usa** los métodos que **ofrece** otra clase para completar su funcionalidad.

```
public class Hello {

      public helloBuddy(String message){
            System.out.println(message);
      }
}
```

Esta clase ofrece el servicio

Esta clase usa el servicio

```
import Hello;

public class SaySomething {

      public static void main(String[] args) {
            Hello hello= new Hello();
            hello.helloBuddy("I say: Hello World!);
      }
}
```

Descripción del problema

- ¿Qué ocurre si las dos clases **no** están en la misma máquina?
- ¿**Dónde** están las clases?
  ¿Cómo se pueden localizar o descubrir las clases con la funcionalidad que se necesita?
- ¿De qué modo se **envía información** entre clases?
  ¿Cómo se "entienden" las clases entre sí y qué protocolo de comunicación siguen?
- ¿Qué sucede si no se encuentra ninguna clase con la funcionalidad necesaria?
  ¿Se puede combinar la funcionalidad de varias clases?

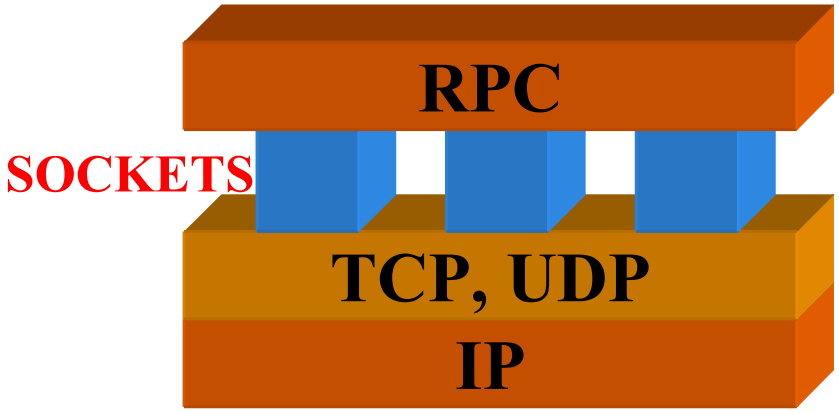
Posibles soluciones

- Soluciones basadas en **llamadas a procedimientos remotos** (RPC).
  - **Distributed Component Object Model (DCOM)**.
  - **Common Object Request Broker Architecture (CORBA)**.
  - **Remote Method Invocation (RMI)**.
- Soluciones basadas en la Web como **una capa software adicional**.
  - **CGIs**.
  - **Servlets**.
  - **Servicios Web**.

RPC

- RPC **esconde** los detalles de comunicación e interacción a la hora de invocar la ejecución de un método que se encuentra en una máquina diferente.
  - El método del cliente hace una invocación de un método que **está implementado** en un servidor.
  - El intercambio de datos tiene lugar a través de los **parámetros de entrada y salida** del método invocado.

- Los métodos se describen en un lenguaje IDL que hace al mecanismo RPC **independiente** del lenguaje de programación.



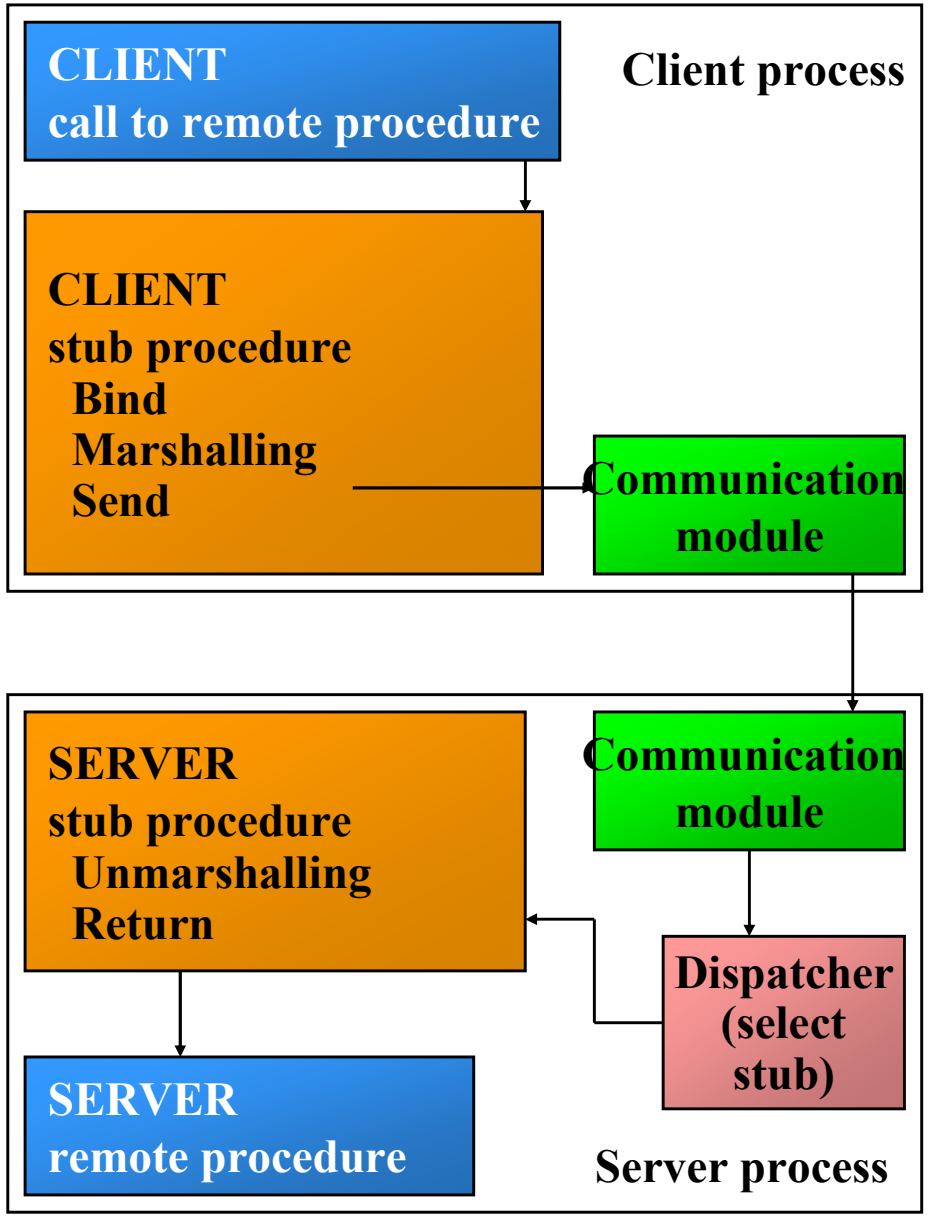
RPC

A través de un compilador se genera **automáticamente** el código del cliente y del servidor que facilita la invocación entre los métodos.

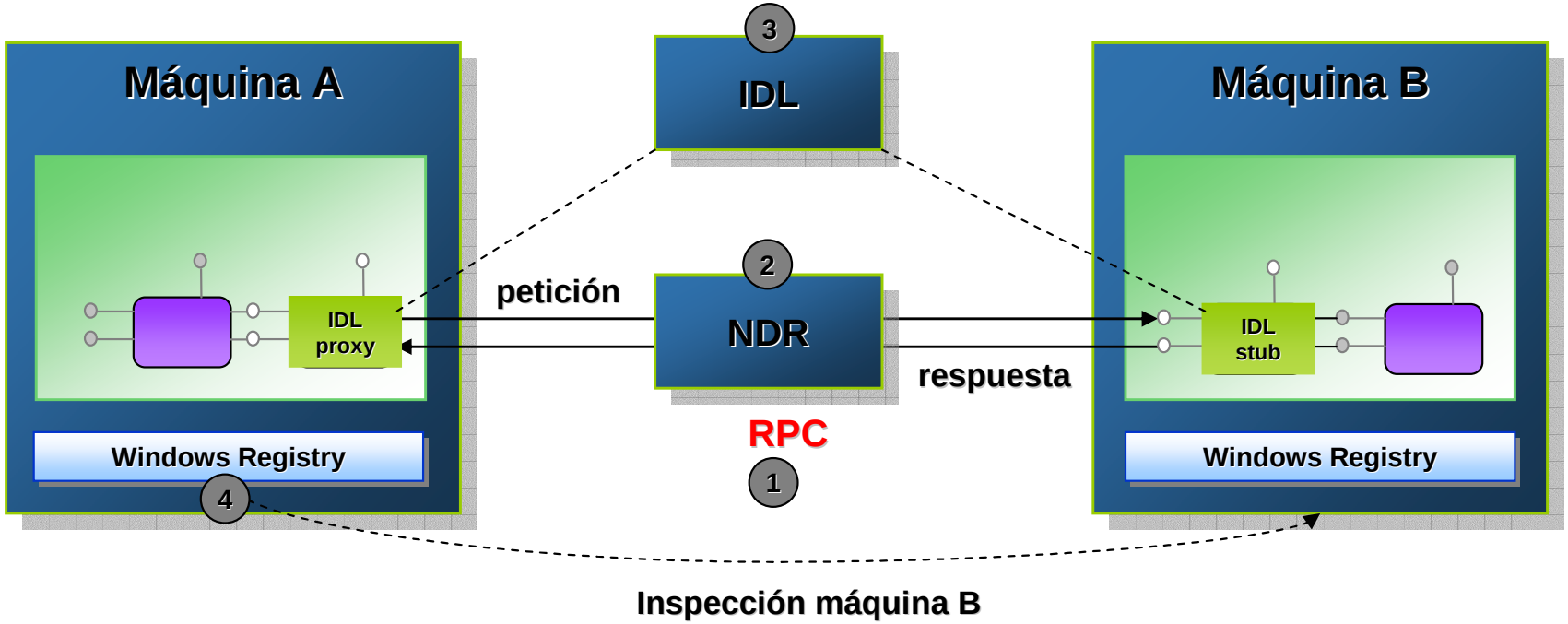
- Operaciones de **codificación** y **decodificación** de parámetros (marshalling / unmarshalling).
- El método del cliente hace una invocación de un método que **está implementado** en un servidor.

Los métodos se dan de alta en un **registro** en el que se les asigna una dirección IP y un puerto.

- Los clientes **conocen de antemano** la definición del método (parámetros de entrada/salida y nombre).

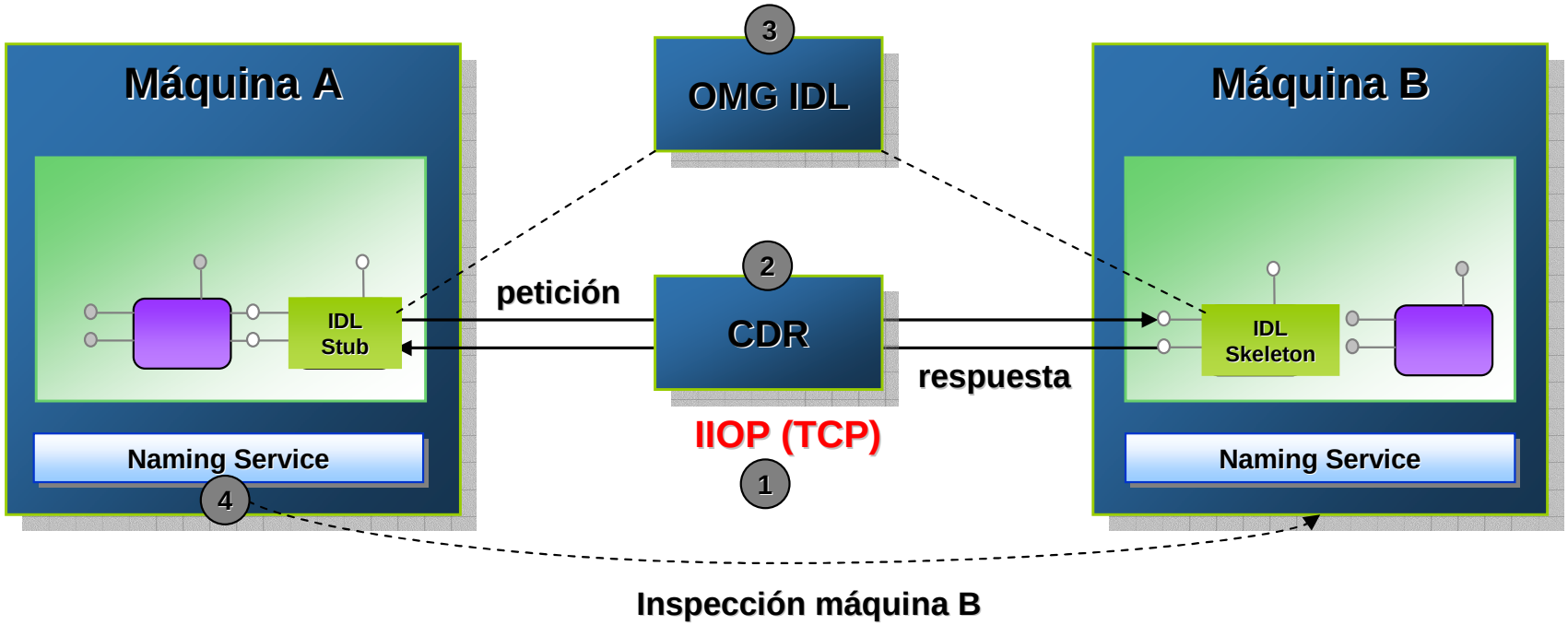


DCOM



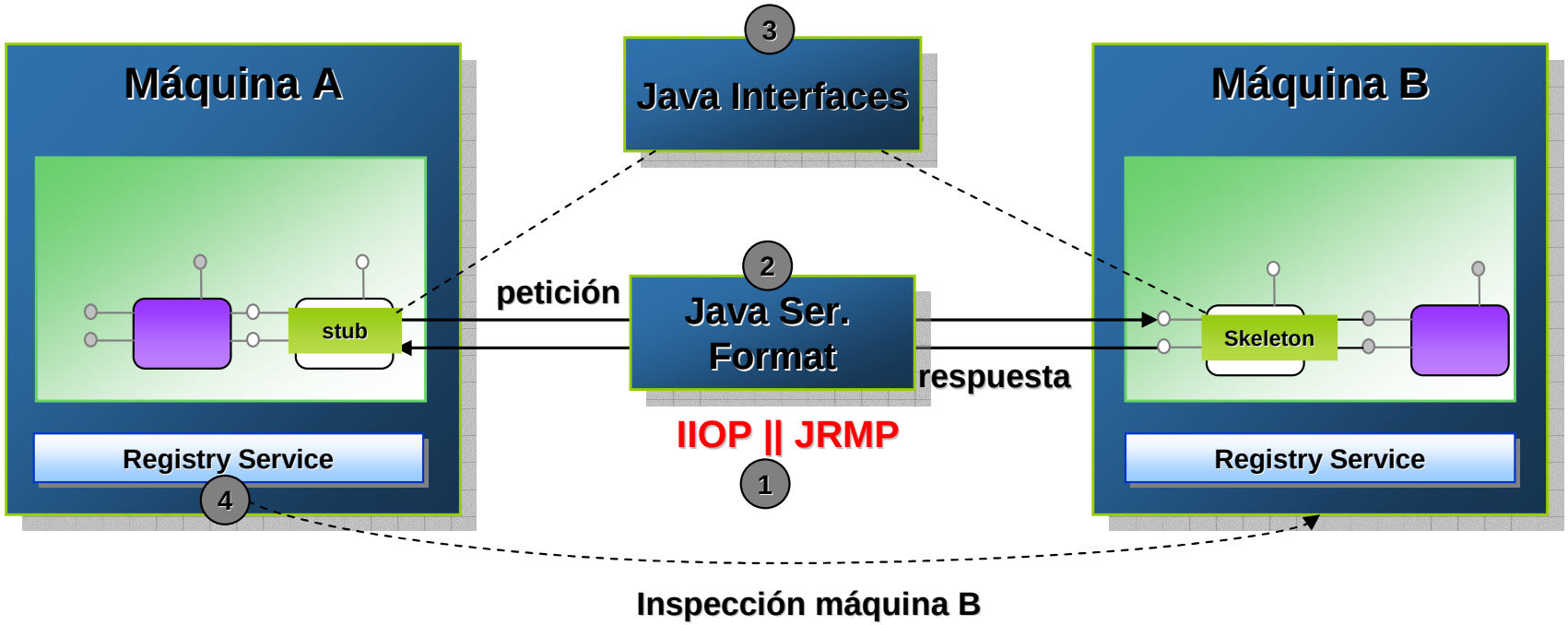
1. Protocolo de comunicaciones
2. Formato de mensaje
3. Lenguaje de descripción
4. Mecanismo de localización

CORBA



1. Protocolo de comunicaciones
2. Formato de mensaje
3. Lenguaje de descripción
4. Mecanismo de localización

RMI



1. Protocolo de comunicaciones
2. Formato de mensaje
3. Lenguaje de descripción
4. Mecanismo de localización

Comparativa entre tecnologías

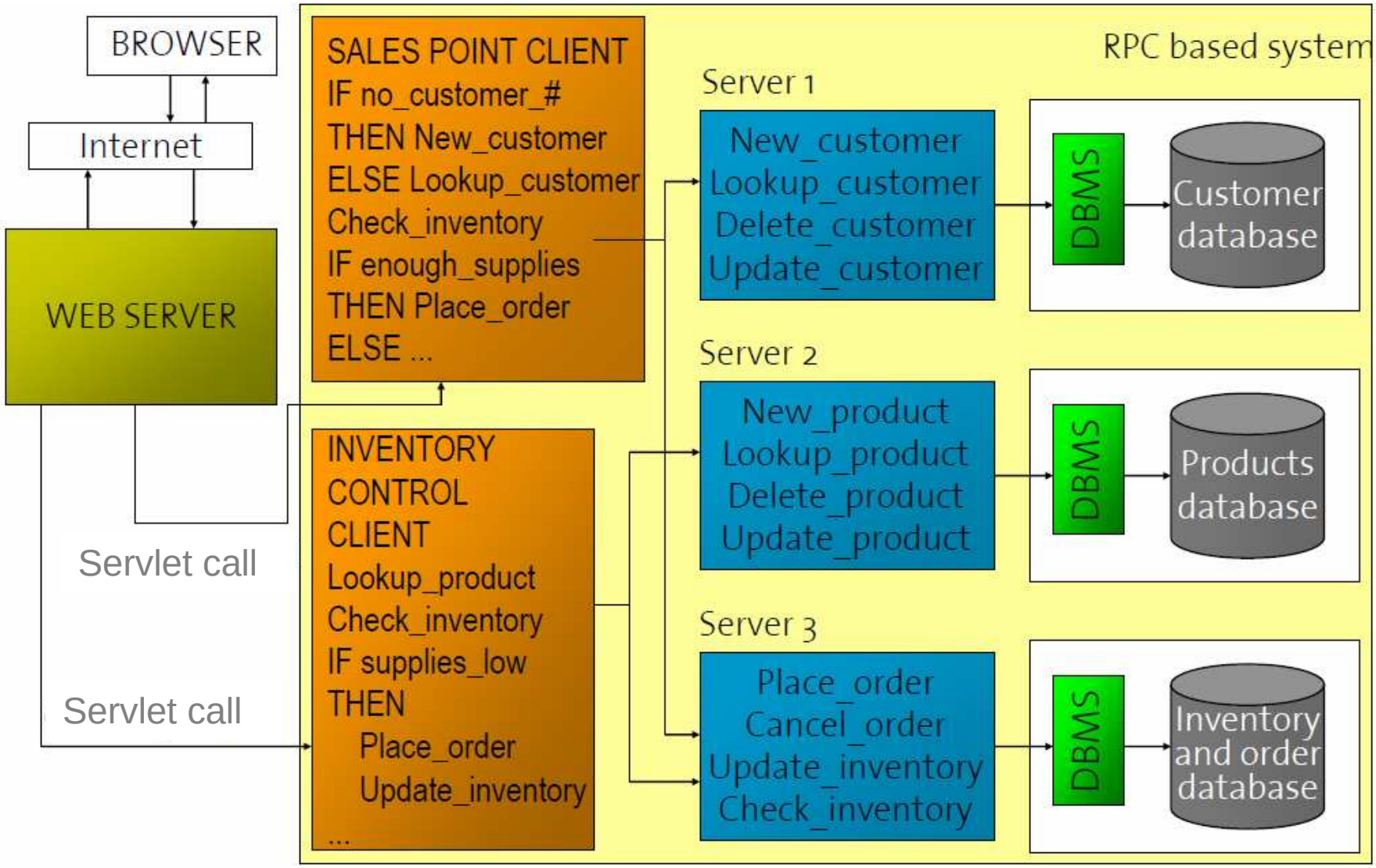
	DCOM	CORBA	Java RMI
RPC Protocol	RPC	IIOP	IIOP or JRMP
Message Format	NDR	CDR	Java Ser. Format
Description	IDL	OMG IDL	Java
Discovery	Windows Registry	Naming Service	RMI Registry or JNDI

PROBLEMAS

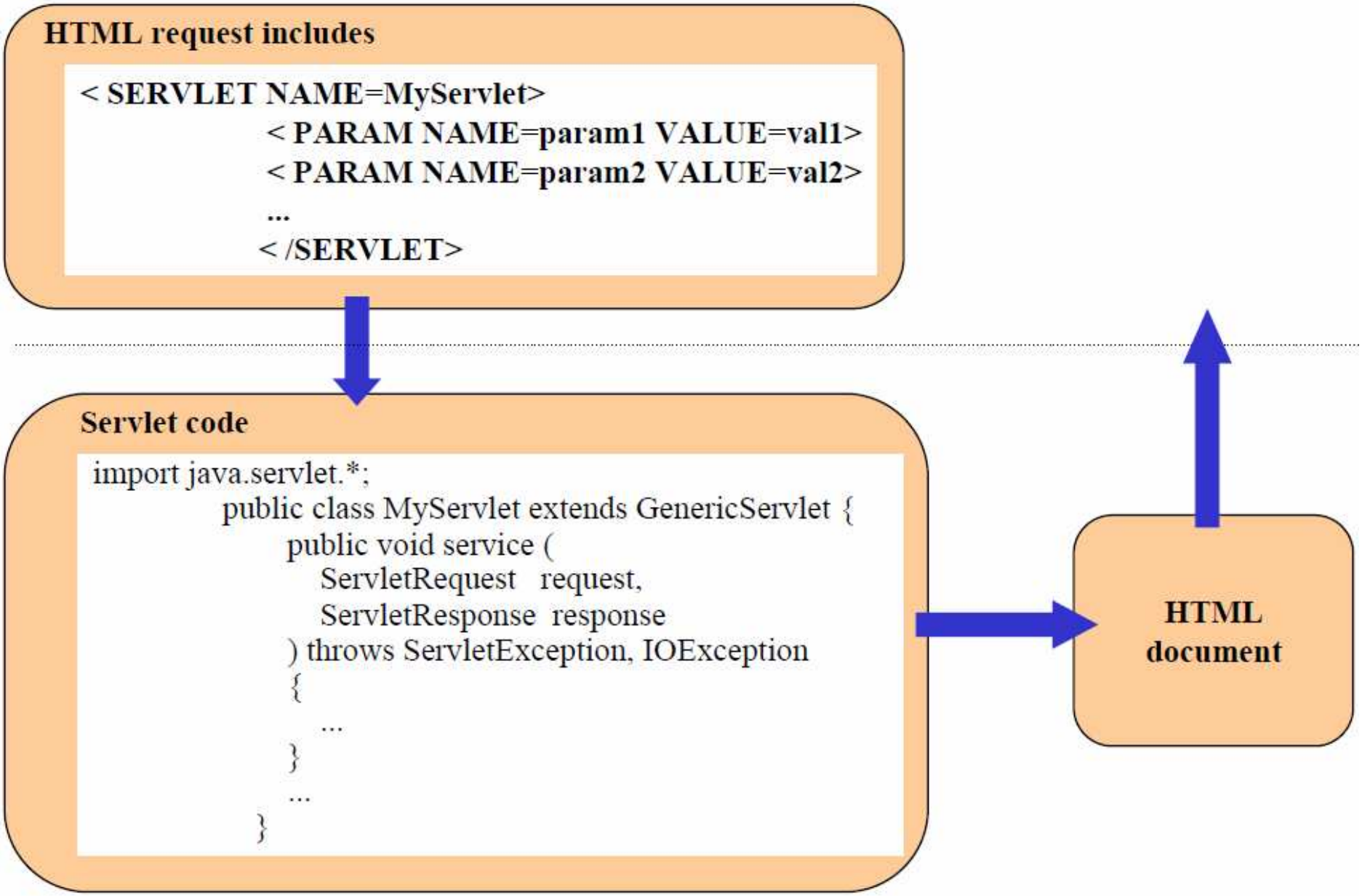
- Estas tecnologías **no interoperan** entre sí.
- Es necesaria una **arquitectura independiente**
  - Del lenguaje,
  - De la plataforma,
  - De las características de los objetos, y
  - Del mecanismo de llamada.

Servlets: Web como capa software

Añade una **capa adicional** a las soluciones basadas en RPC (**servidor de aplicaciones**)



Servlets: Web como capa software



Servlets: Problemas

- La interacción entre los métodos tiene lugar a través de documentos o cadenas de texto.
  - **No se manejan tipos de datos.**
- Los métodos que se pueden invocar **están predefinidos** (POST y GET) y no es posible indicar parámetros de entrada/salida que no sean cadenas de texto.
  - La ejecución de otros métodos requiere de un parámetro de entrada en el que vaya codificado un lenguaje de invocación de los métodos.
    (solución **ad hoc**)
  - **No es lo suficientemente flexible** para la integración de aplicaciones o para la reutilización de la funcionalidad ofrecida por el servlet.

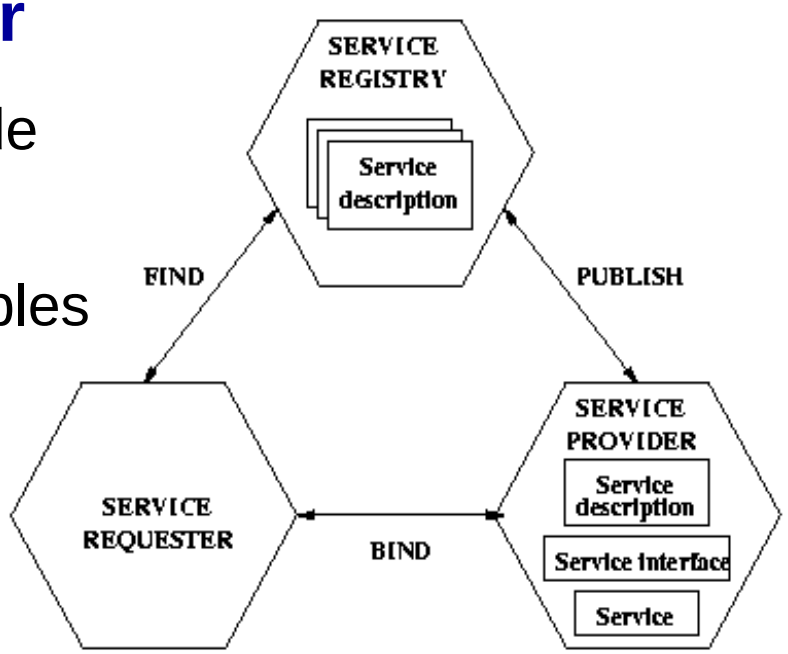
Solución: Arquitectura SOA

- El término arquitectura orientada a servicio (SOA) **no está ligado** a ninguna tecnología o a lenguajes de descripción de protocolos de interacción y de componentes.
  - **Una arquitectura SOA no tiene por qué estar implementada con servicios web.**
  - **No todo servicio es un servicio web.**

- Elementos de una SOA: **Servicios**
  - son componentes software **bajamente acoplados** (***loosely decoupled***) por lo que se podrían ser:
  - Reutilizados por otros componentes de la misma arquitectura. (**independientes**)
  - **Combinados entre sí** para proporcionar las funcionalidades requeridas por los clientes. (**orquestación**)

Solución: Arquitectura SOA

- Elementos de una SOA: **Proveedor**
  - Componente que **ofrece** un conjunto de servicios con una funcionalidad dada.
  - Los servicios son directamente accesibles a través de Internet, es decir, **están expuestos a través de URLs**.
  - Utiliza un lenguaje de **descripción** estándar de servicios.



- Elementos de una SOA: **Consumidor**
  - Componente que **invoca o consume** la funcionalidad de los servicios ofrecidos por el proveedor.
  - Utiliza un protocolo de **invocación**.
- Elementos de una SOA: **Registro**
  - Componente que **contiene** los servicios ofrecidos por el proveedor.

Solución: Arquitectura SOA / Servicios Web

- Los **servicios web son interfaces** que describen las características de una colección de operaciones (o métodos).
  - que son **accesibles a través de la red** usando protocolos Web **estandarizados** que están basados en formatos XML.
    (**invocación**)
  - y cuyas propiedades están representadas usando un lenguaje **estándar** basado en un formato XML.
    (**descripción**)
- Los formatos XML describen la **estructura** de los protocolos de invocación y de la descripción de los servicios.

Solución: Arquitectura SOA / Servicios Web

- La institución que se encarga de estandarizar los lenguajes y protocolos en la Web es el **Consorcio W3C**.

- En la tecnología de servicios Web se hace uso de los siguientes estándares que están representados en XML:

  - **Protocolo de comunicaciones: HTTP**
    (http://www.w3.org/Protocols)
  - **Formato del mensaje: SOAP**
    (http://www.w3.org/TR/soap)
  - **Descripción de los servicios: WSDL**
    (http://www.w3.org/TR/wsdl)

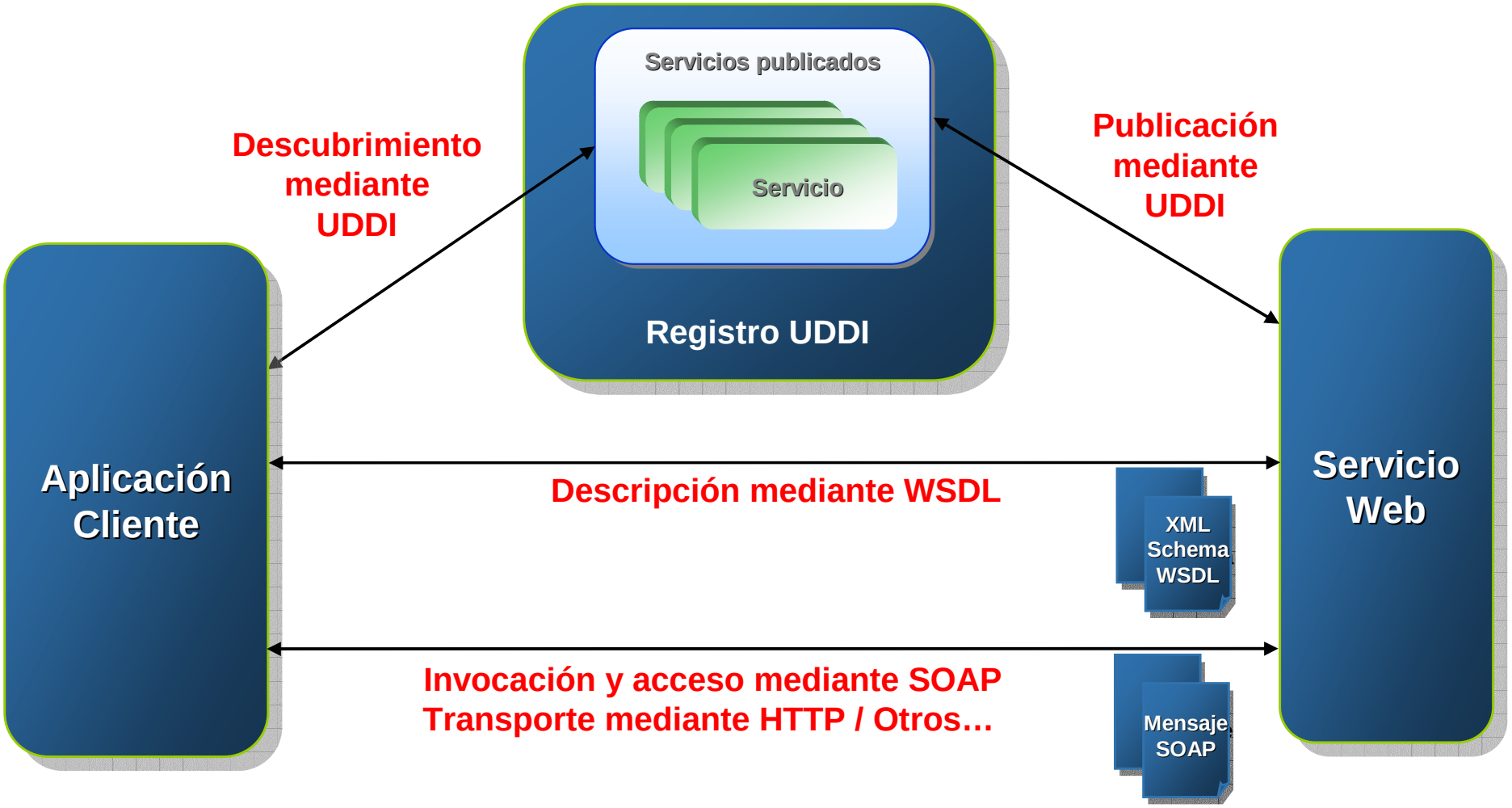
  Estándares del **Consorcio W3C**

  - **Protocolo y registro: UDDI**
    (http://www.uddi.org)

Solución: Arquitectura SOA / Servicios Web

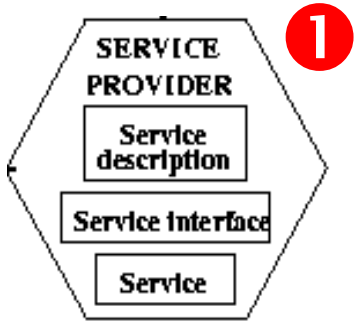
La arquitectura SOA "recuerda" a las soluciones basadas en la tecnología RPC

Las principales diferencias se encuentran en la **estandarización de protocolos** y en **el énfasis en la composición de servicios**.



Solución: Arquitectura SOA / Servicios Web

❶ El proveedor **despliega** el conjunto de operaciones que desea hacer accesibles a través de Internet.



– Son accesibles como direcciones URL que apuntan a recursos (ficheros) descritos en el **lenguaje WSDL**.

– Con WSDL solamente se describen las capacidades **funcionales** de cada una de las operaciones. Es decir:

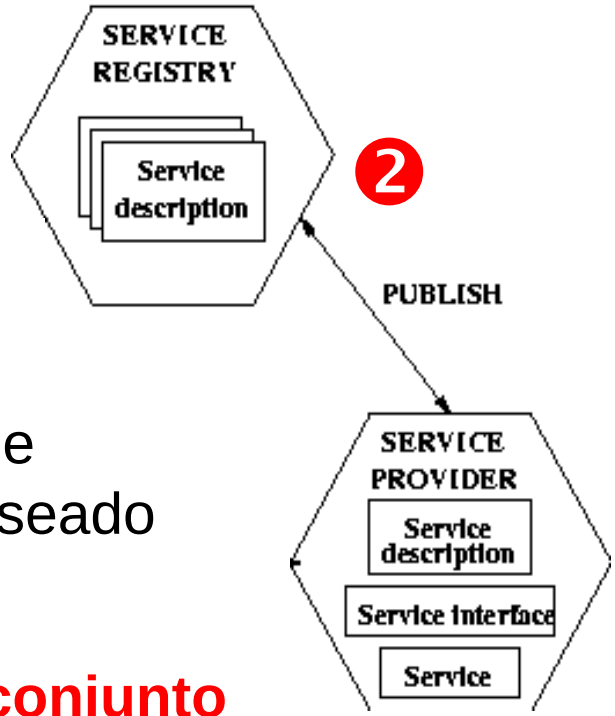
> WSDL contiene interfaces en los que se indica el **nombre**, las **entradas**, y las **salidas** de las operaciones.
>
> WSDL también indica de **qué modo se invocarán** las operaciones.

– ¿Estas operaciones son **realmente accesibles** por parte de los programas cliente?

> **Lo serán si el cliente conoce las URLs.**

Solución: Arquitectura SOA / Servicios Web

❷ El proveedor **publica** a través del protocolo UDDI las características de los servicios en un registro que puede ser consultado por los clientes (**registro UDDI**):



– En el registro se indican las características **no funcionales** de los servicios, tales como la descripción de la empresa que los ofrece, la categoría a la que pertenecen, etc.

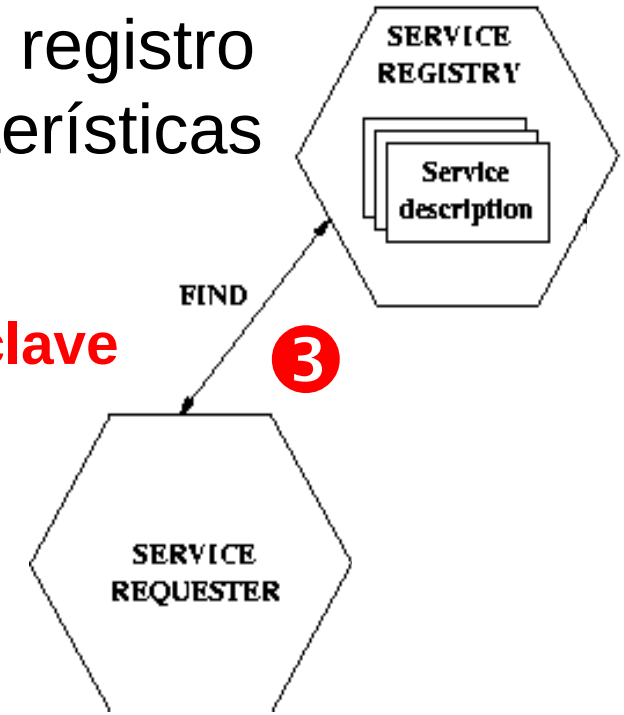
– En el registro también **se indica la URL** correspondiente al fichero WSDL que contiene las características funcionales del servicio deseado por el cliente.

– Para acceder al registro UDDI se utilizan un **conjunto de APIs** que permiten dar de alta y de baja los servicios que ofrece el proveedor.

– Un registro UDDI es a los servicios Web lo que los **DNS** a las direcciones Web.

Solución: Arquitectura SOA / Servicios Web

❸ El consumidor **busca** los servicios en el registro UDDI los servicios que tienen las características deseadas:



- La búsqueda se realiza indicando **palabras clave** que hacen referencia a las características funcionales y no funcionales de los servicios.
- Se pueden indicar las **entradas y salidas** que deben tener los servicios (TModel).
- Como resultado de la búsqueda se devuelve una **URL apuntando al fichero WSDL** que contiene la descripción de las operaciones y la forma en la que pueden ser invocadas.

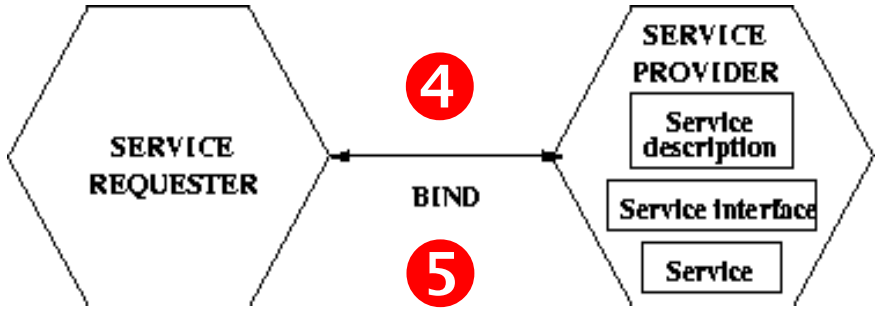
UDDI **no es un estándar** del Consorcio W3C por lo que es uno de los componentes de menor implantación en el mercado.

Solución: Arquitectura SOA / Servicios Web

❹ El consumidor obtiene el fichero WSDL y **genera automáticamente** el código necesario para realizar la invocación de las operaciones descritas.

- Se generan las clases que representan **los tipos de datos** de los parámetros de entrada/salida.
- Se generan las clases que **codifican** y **decodifican** los parámetros de entrada/salida de las operaciones.

❺ El consumidor usa el **protocolo SOAP para invocar** la ejecución de una de las operaciones definidas en el fichero WSDL que ha sido localizado en el registro UDDI.



Ventajas: Arquitectura SOA / Servicios Web

- El uso de formatos XML para describir e invocar las operaciones **permite la integración** de aplicaciones que:
  - Están implementadas en lenguajes de programación **diferentes**.
    (Traducción del formato XML al lenguaje de programación)
  - Utilizan modelos de datos **diferentes**.
    (Traducción del modelo representado en el formato XML al modelo de la aplicación)
- El uso de servicios como **componentes bajamente acoplados** permite a los consumidores:
  - **Reutilizar** las operaciones en diferentes aplicaciones.
  - **Combinar** las operaciones entre sí para obtener nuevas funcionalidades.