

# Aplicaciones de Internet: SOAP

Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid



# Contenidos

---

## 1. Introducción:

1. Paradigma de servicios de red

## 2. SOAP

1. Introducción
2. Arquitectura
3. Ejemplo de aplicación
  - ▶ Desarrollo de un servicio privado

# Contenidos

---

## 1. **Introducción:**

### 1. **Paradigma de servicios de red**

## 2. SOAP

### 1. Introducción

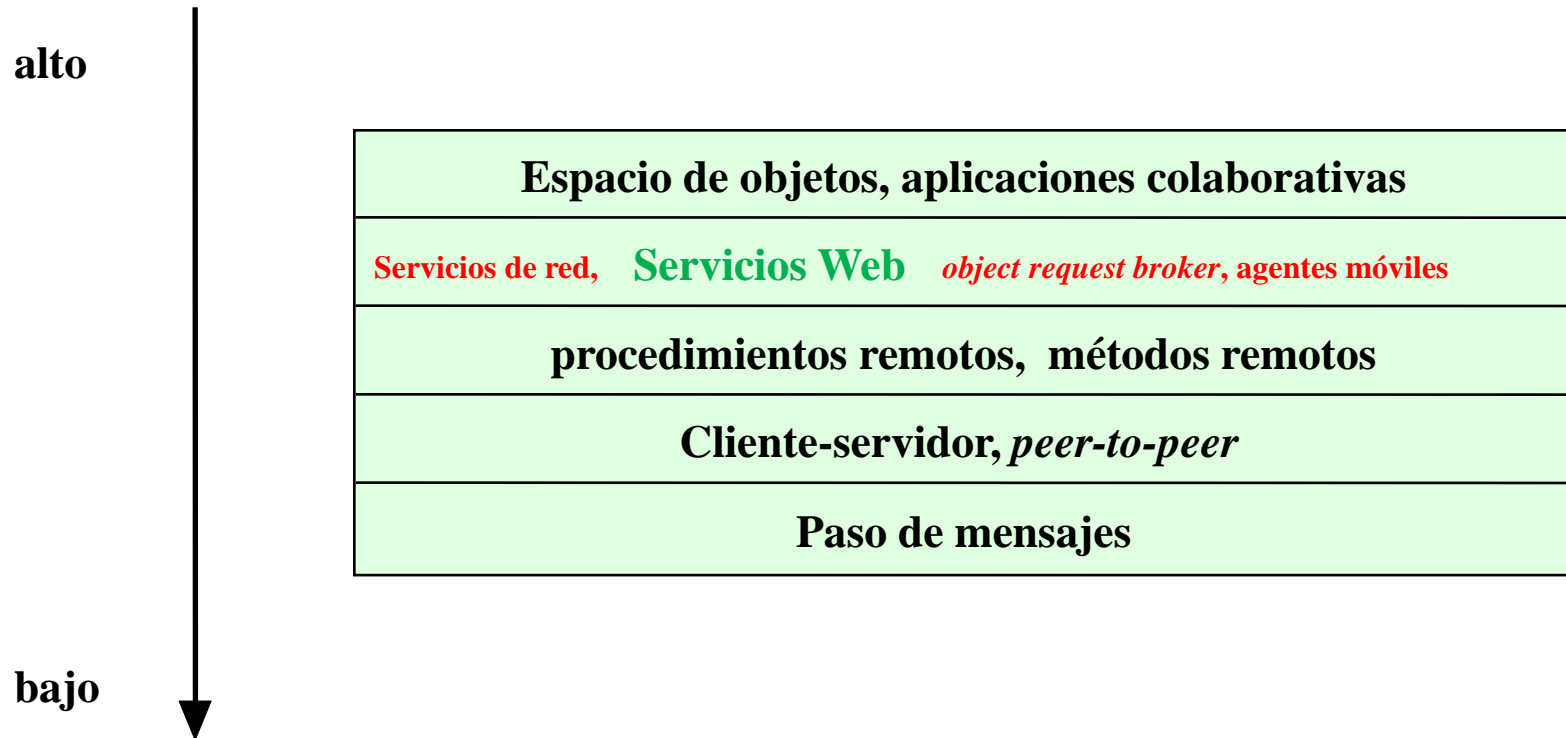
### 2. Arquitectura

### 3. Ejemplo de aplicación

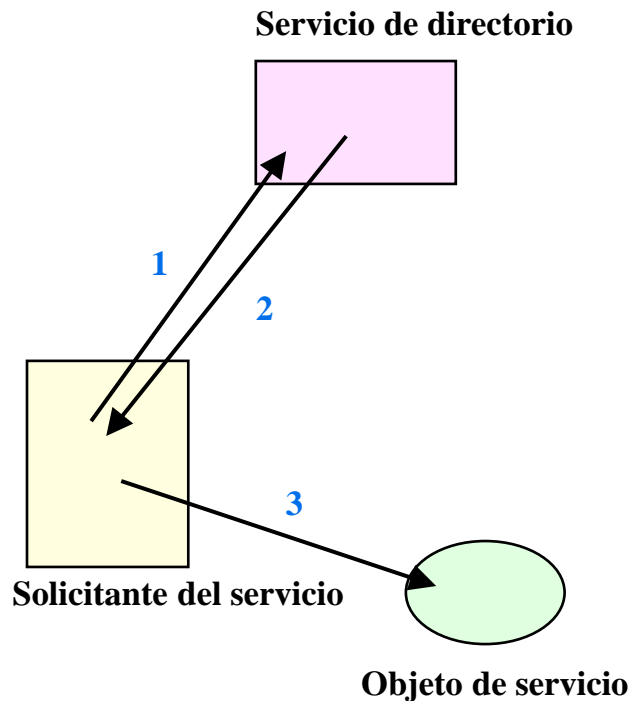
- ▶ Desarrollo de un servicio privado

# Paradigmas de Servicios de red, ORB, etc.

---



# 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

# Contenidos

---

## 1. Introducción:

1. Paradigma de servicios de red

## 2. **SOAP**

### 1. **Introducción**

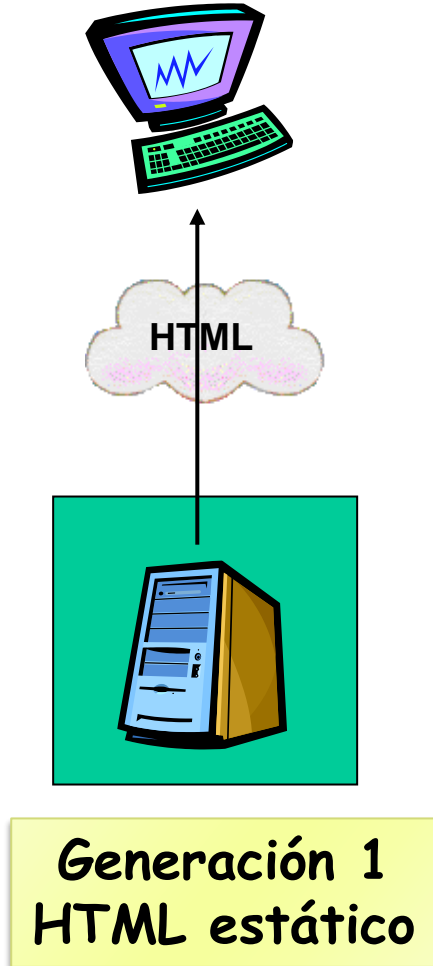
### 2. Arquitectura

### 3. Ejemplo de aplicación

- ▶ Desarrollo de un servicio privado

# Evolución de la Web...

---



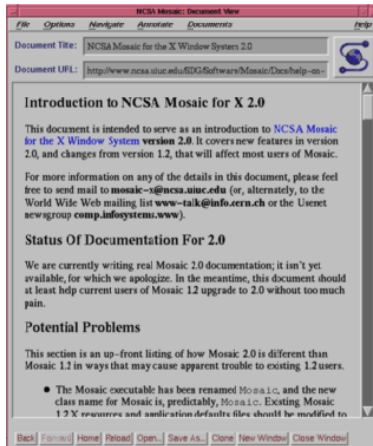
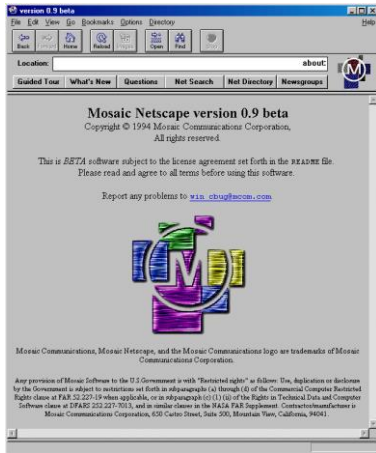
- ▶ El navegador Web pide una página Web indicando su identificador URI en la petición.
- ▶ El servidor Web busca el fichero almacenado que se corresponde con la URI pedida, y lo envía como respuesta.
- ▶ Se utiliza el protocolo HTTP para la transferencia de contenido.
- ▶ Contenido diverso:
  - ▶ Páginas HTML
  - ▶ Imágenes: PNG, JPEG, etc.
  - ▶ Vídeos: mov, AVI, etc.
  - ▶ Sonidos: MP3, .wav, etc.



# Ejemplo de la generación 1



# Ejemplo de la generación 1



VMS Mosaic 4.3 [Main Page - Wikipedia, the free encyclopedia]

File Options Navigate Annotate News Documents Debug Help

URL: [http://en.wikipedia.org/wiki/Main\\_Page](http://en.wikipedia.org/wiki/Main_Page)

## Main Page

From Wikipedia, the free encyclopedia


Jump to: [navigation](#), [search](#)

Welcome to [Wikipedia](#), the free [encyclopedia](#) that [anyone can edit](#).  
[2,188,059](#) articles in [English](#)

- [Arts](#)
- [Biography](#)
- [Geography](#)
- [History](#)
- [Mathematics](#)
- [Science](#)
- [Society](#)
- [Technology](#)
- [All portals](#)


[Overview](#) · [Editing](#) · [Questions](#) · [Help](#) [Contents](#) · [Categories](#) · [Featured content](#) · [A-Z index](#)

### Today's featured article



**Stede Bonnet** was an early 18th-century Barbadian [pirate](#), sometimes called "the gentleman pirate". Because of marital problems, Bonnet turned to piracy in the summer of 1717. He bought a sailing vessel, named it *Revenge*, and traveled with his paid crew along the American eastern seaboard, capturing other vessels and burning down Barbadian ships. After arriving in [Nassau](#), Bonnet met the infamous pirate [Blackbeard](#). Incapable of leading his crew, Bonnet temporarily ceded his ship's command to Blackbeard. Before separating in December 1717, Blackbeard and Bonnet plundered and captured merchant ships along the East Coast. After Bonnet failed to capture the *Protestant Caesar*, his crew abandoned him to join Blackbeard on the *Queen Anne's Revenge*. Bonnet stayed on Blackbeard's

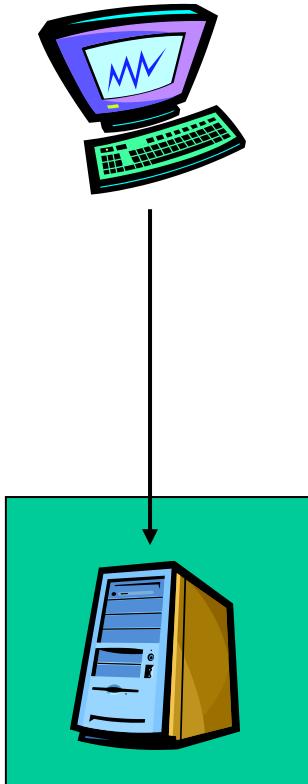
### In the news



- A peace deal ends the [Kivu conflict](#) in the [Democratic Republic of the Congo](#).
- [Stock markets](#) throughout the world [fall](#), resulting in the [U.S. Federal Reserve](#) cutting [interest rates](#) by 0.75%, the largest single cut in interest rates since October 1984.
- [Tomislav Nikolić](#) and [Boris Tadić](#) advance to the second round of [presidential elections](#) in [Serbia](#).
- [British Airways Flight 38](#) (pictured) crash

100%

# Ejemplo de la generación 1: HTTP 1.0

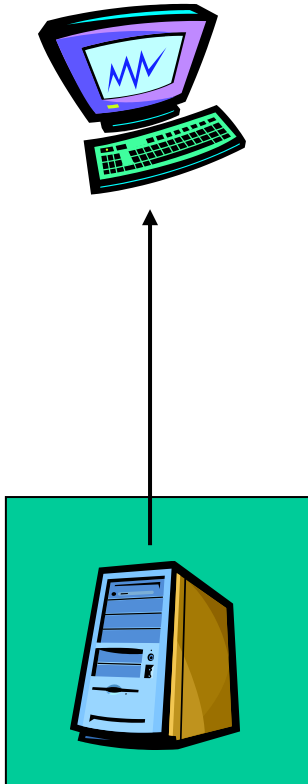


```
telnet www.uc3m.es 80
```

```
GET / HTTP/1.0  
Accept: */*  
Host: www.inf.uc3m.es  
User-Agent: firefox
```



# Ejemplo de la generación 1: HTTP 1.0



```
HTTP/1.1 200 OK
```

```
Date: Sat, 15 Sep 2001 06:55:30 GMT
```

```
Server: Apache/1.3.9 (Unix)
```

```
ApacheJServ/1.0
```

```
Last-Modified: Mon, 30 Apr 2001
```

```
23:02:36 GMT
```

```
ETag: "5b381-ec-3aedef0c"
```

```
Accept-Ranges: bytes
```

```
Content-Length: 236
```

```
Connection: close
```

```
Content-Type: text/html
```

```
<html>
```

```
<head>
```

```
<title>My web page </title>
```

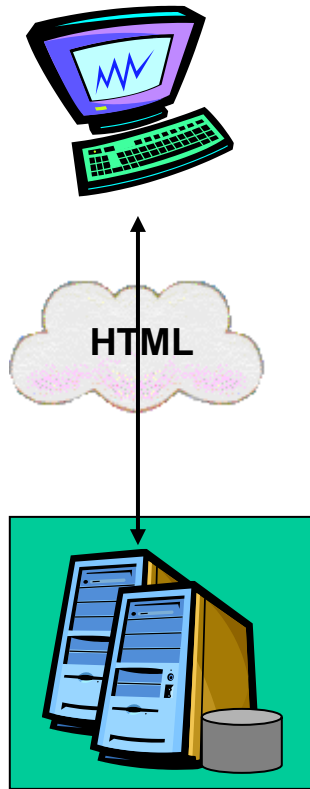
```
</head>
```

```
<body>
```

```
Hello world!
```

```
</BODY></HTML>
```

# Evolución de la Web...



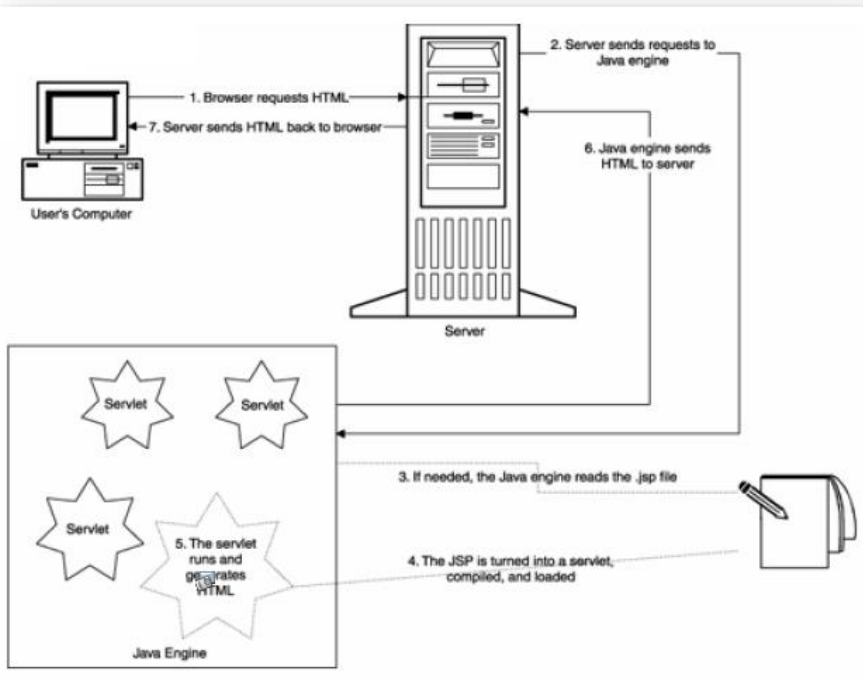
## Generación 2 Aplicaciones Web

- ▶ Se añade la posibilidad de enviar datos al servidor (POST o GET) a través de formularios.
- ▶ Dos estrategias:
  - ▶ En el servidor:
    - ▶ Ejecución de programa en el servidor al que se le pasa los datos del formulario, y cuya salida se envía al cliente: CGI, *servlets* de Java, lenguajes embebidos (PHP, JSP, ASP, etc.)
  - ▶ En el cliente:
    - ▶ Además de páginas, imágenes, videos, etc. transferencia de aplicaciones para el navegador Web: *applets* de Java, flash, Adobe AIR, Microsoft Silverlight, etc.
    - ▶ Ejecución en el navegador Web del cliente de ciertas operaciones (libera al servidor de parte de la carga)

# Ejemplo de la generación 2



# Ejemplo de la generación 2: Tomcat



- ▶ Tomcat implementa las especificaciones de *Servlets* y *JSP*. Además incluye un servidor **HTTP**.
  - ▶ Tomcat es un servidor Web y contenedor de *servlets* y *JSP*
- ▶ Programado en *java* desarrollado por *Apache Software Foundation* bajo el proyecto *Apache Jakarta*.
- ▶ Un contenedor de *servlets* consiste esencialmente de una **aplicación que hace de anfitriona de los java servlets**:
  - ▶ El contenedor controla el *servlets* que esta ejecutando dentro del servidor web
  - ▶ Mapea la dirección URL a un *servlet* en particular y asegurarse que el proceso de requerimientos de direcciones tenga los permisos adecuados.
  - ▶ Es responsable de retransmitir las peticiones y respuestas que le hacen al *servlet*.

# Ejemplo de la generación 2: *Servlet*

---

- ▶ La palabra *servlet* se deriva de la anterior *applet*:
  - ▶ Un *applet* es un programa en Java que se ejecutan en el *navegador Web*.
  - ▶ Un *servlet* es un programa que se ejecuta en un *servidor Web*.
- ▶ Un *servlet* **permite generar páginas Web dinámicas** a partir de los parámetros de la petición que envíe el navegador web.
- ▶ Los *servlets* **forman parte de J2EE** (*Java 2 Enterprise Edition*), que es una ampliación de J2SE (*Java 2 Standard Edition*).
- ▶ Un *servlet* **es un objeto Java que implementa** la interfaz `javax.servlet.Servlet` o **hereda** para algún protocolo específico (ej: `javax.servlet.HttpServlet`).
- ▶ Un *servlet* es un objeto que **se ejecuta en un servidor o contenedor J2EE**.



# Ejemplo de la generación 2: *Servlet*

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletHolaMundo extends HttpServlet {
    public void doGet (HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>"+
            "<head><title>Hola Mundo con un servlet</title></head>"+
            "<body><div align='center'><b>Hola Mundo </b></div>"+
            "</body></html>");
    }

    public void doPost (HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {
        doGet(request, response);
    }
}
```

# Ejemplo de la generación 2: JSP

---

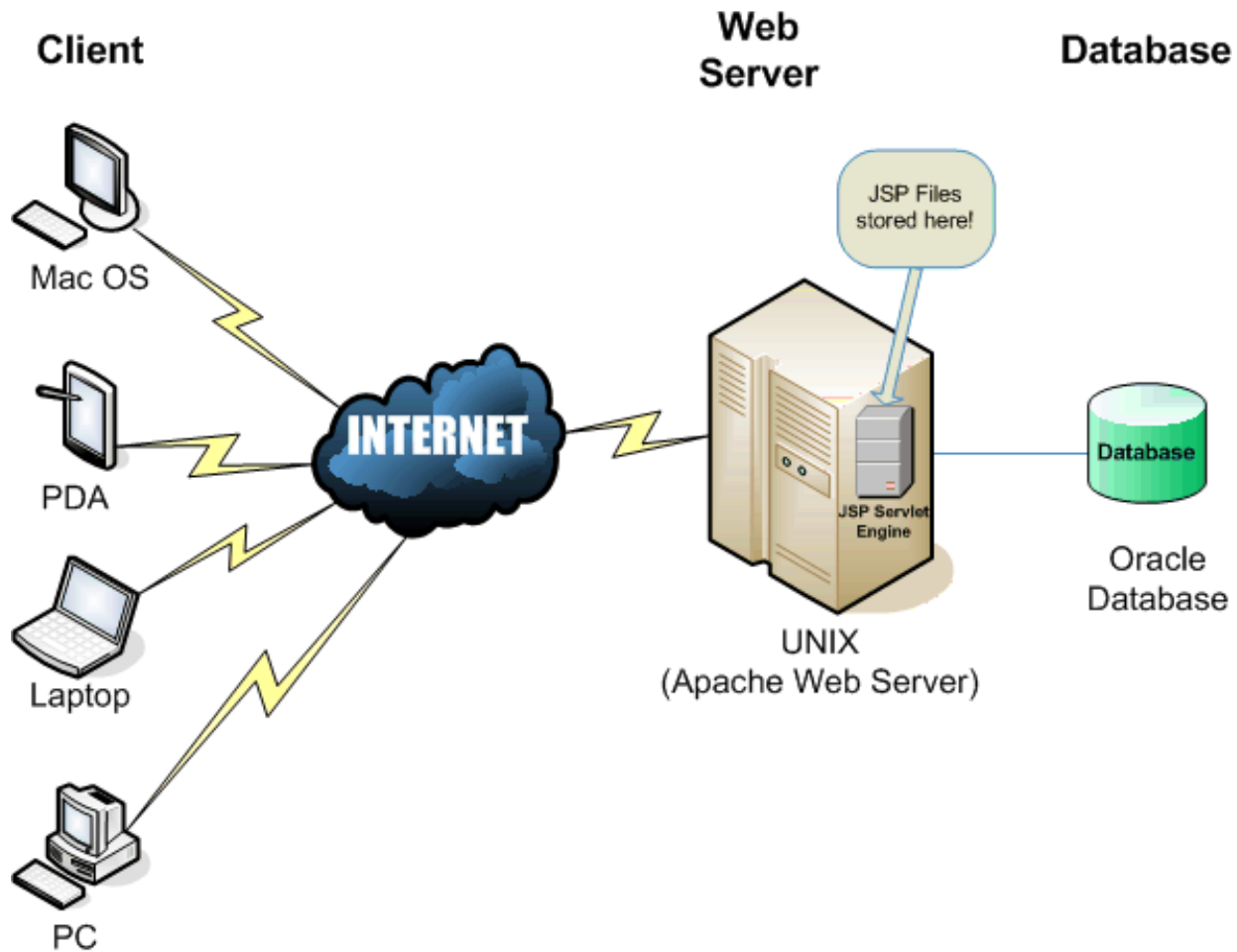
- ▶ **Permite combinar** código **HTML** estático con código generado en un mismo fichero.
  - ▶ Los JSP nos permiten separar la parte dinámica de nuestras páginas Web del HTML estático
  - ▶ Simplemente **escribimos el HTML** cómo habitualmente **y encerramos** el **código** de las **partes dinámicas** en unas **etiquetas** especiales, la mayoría de las cuales empiezan con "**<%**" y terminan con "**%>**"
  - ▶ Aunque el código parezca más bien HTML, el **servidor lo traduce a un *servlet* en la primera petición**
- ▶ **JSP vs ASP:**
  - ▶ Respuesta de Sun Microsystems a ASP e Microsoft
- ▶ **JSP vs Servlet:**
  - ▶ Los JSP son interpretados en *servlet*
  - ▶ JSP es una extensión de los *servlets*
  - ▶ Código más limpio
  - ▶ Separación de presentación e implementación

# Ejemplo de la generación 2: *JSP*

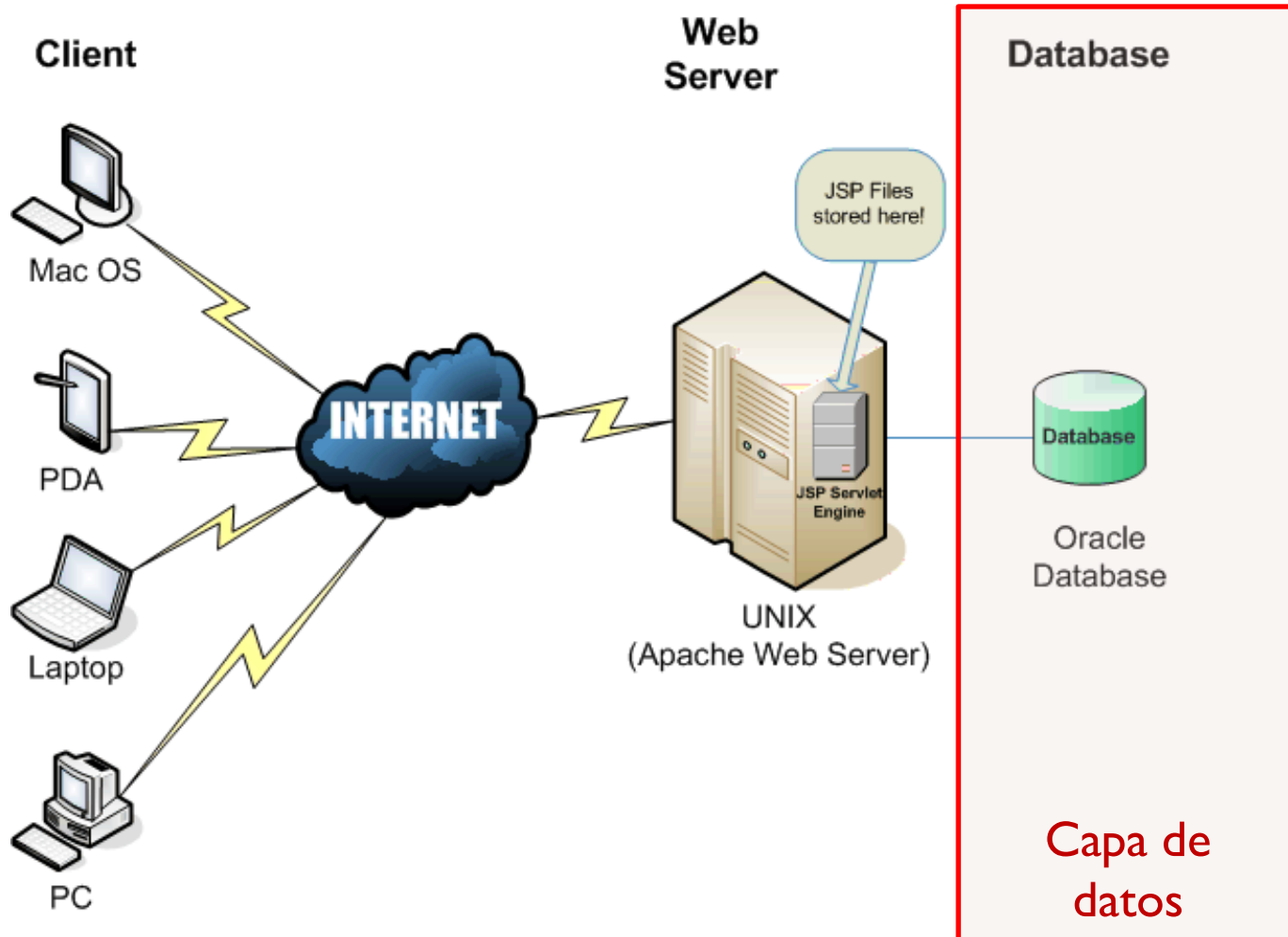
---

```
<%@ page language="Java"%>
<html>
  <head>
    <title>Hola mundo con JSP</title>
  </head>
  <body>
    <!--Esto es un comentario-->
    <div align="center">
      <b><%out.println("Hola Mundo");%></b>
    </div>
  </body>
</html>
```

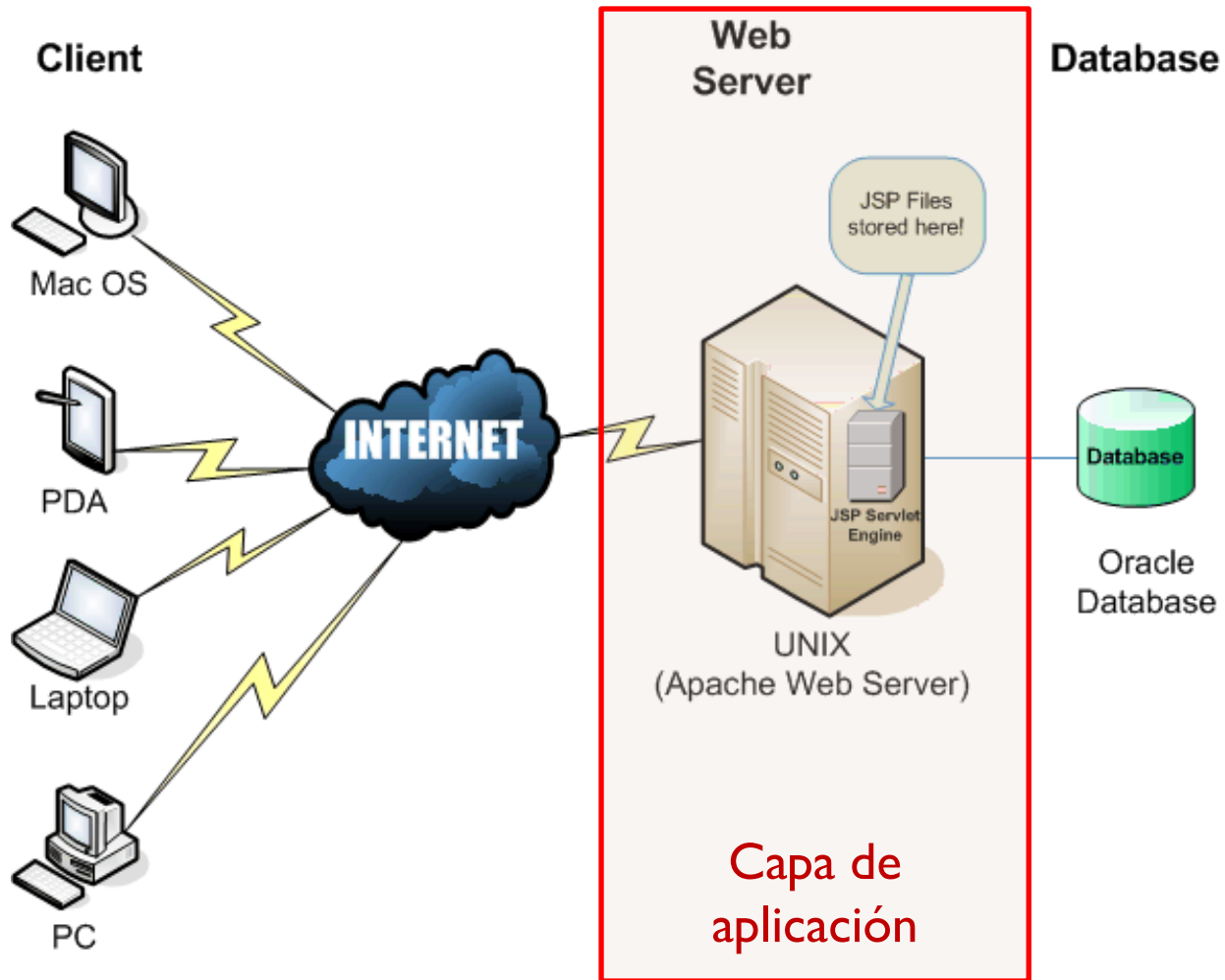
# Arquitectura en tres capas (3-tier)



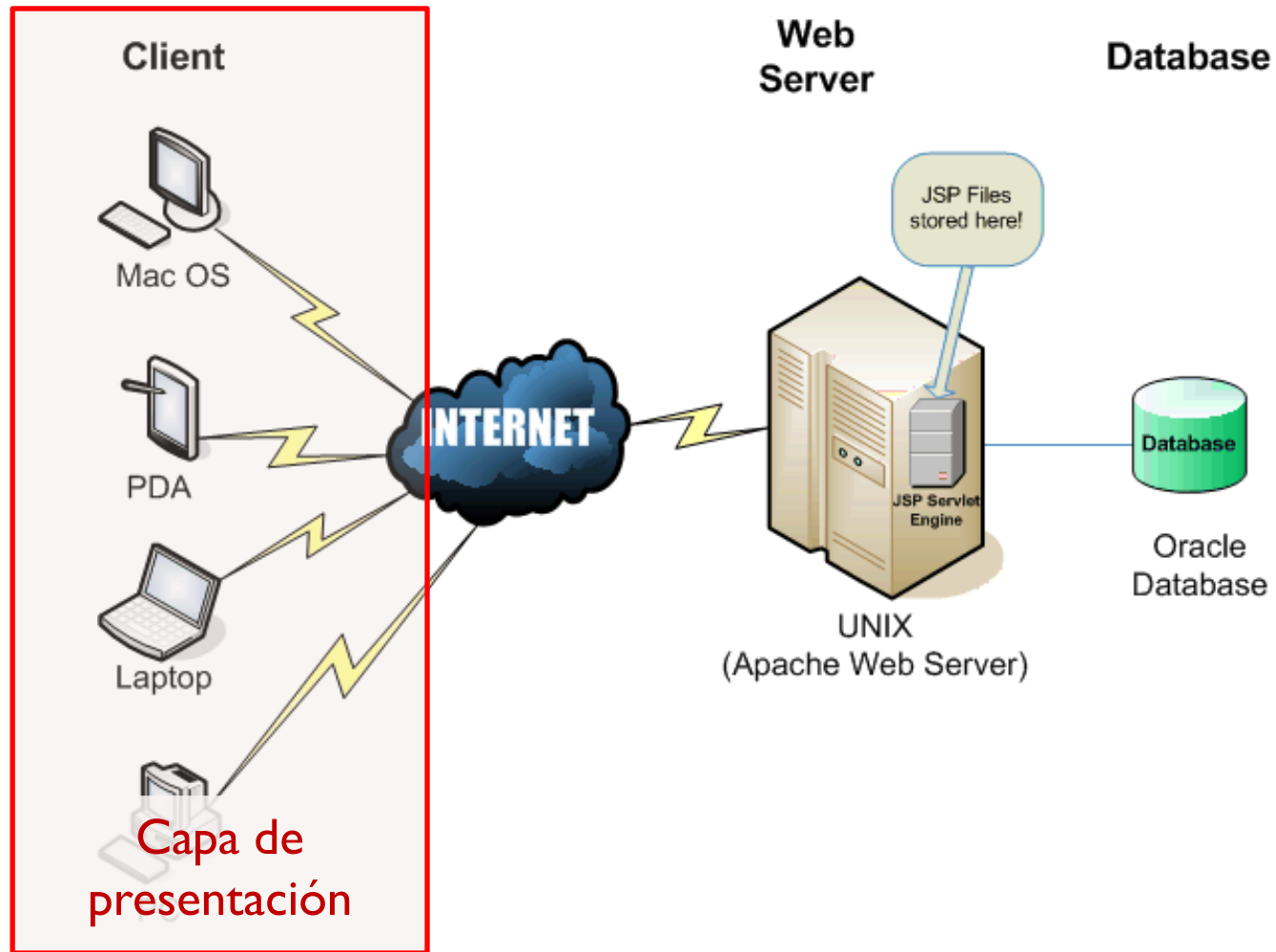
# Arquitectura en tres capas (3-tier)



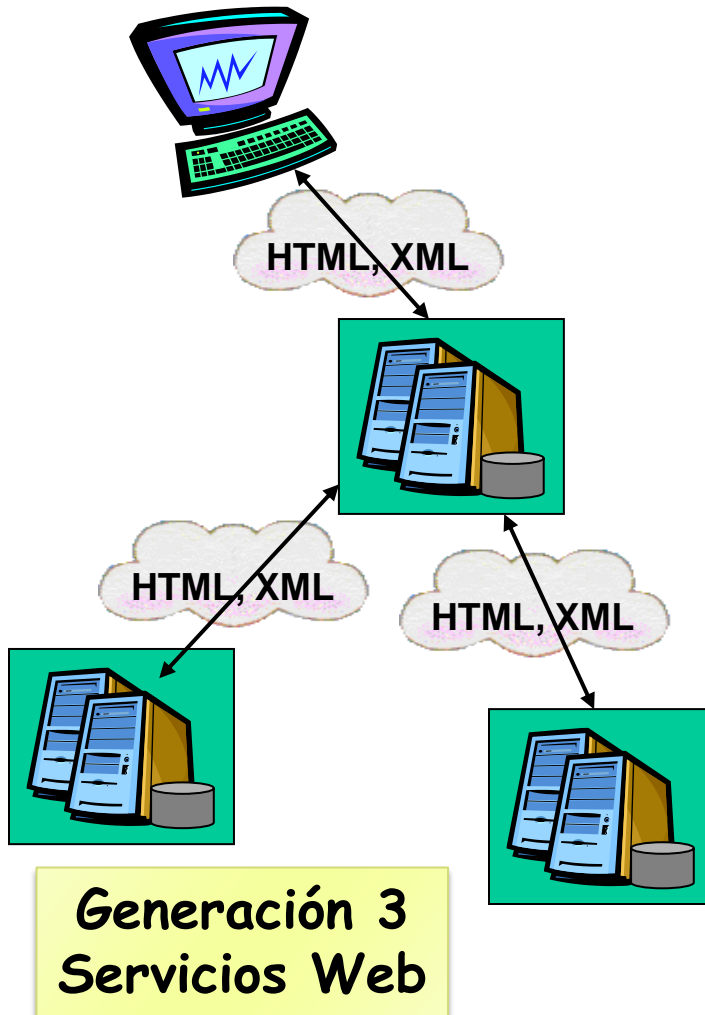
# Arquitectura en tres capas (3-tier)



# Arquitectura en tres capas (3-tier)



# Evolución de la Web...



- ▶ Aparece b2b (*business to business*)
  - ▶ Necesidad de comunicar procesos de empresas sobre internet
    - ▶ Ej.: agencia de viaje que reserva avión y hotel
- ▶ Problema de la segunda generación:
  - ▶ Muy diversas tecnologías:
    - ▶ *Applets*, CGI, Lenguajes de *Scripts*, etc.
  - ▶ Desarrollos **muy centrados** en la **interacción con la persona**.
  - ▶ Por seguridad, los **cortafuegos** (*firewalls*) de muchas empresas **solo** dejan pasar **tráfico HTTP** (puerto 80) y cierran el resto:
    - ▶ Dificultad para usar Java RMI o CORBA
- ▶ Tercera generación: servicios Web



# Servicio Web

---



- ▶ Un **servicio web** (en inglés, **Web Service**) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones en redes de ordenadores como Internet.
  - ▶ Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos.
  - ▶ La interoperabilidad se consigue mediante la adopción de estándares abiertos.
- ▶ Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web.

# Servicio Web

---

- ▶ Principales protocolos usados:
  - ▶ HTTP: transporte utilizado
  - ▶ SOAP: empaqueta la información y la transmite entre el cliente y el proveedor del servicio
  - ▶ XML: describe la información, los mensajes
  - ▶ UDDI: lista de servicios disponibles
  - ▶ WSDL: descripción del servicio



# Servicio Web

---



- ▶ **Ventajas:**

- ▶ **Paso de cortafuegos**

- ▶ Difícil en otros entornos como Java RMI o CORBA

- ▶ **Interoperabilidad**

- ▶ **Compatibilidad**

- ▶ Especificaciones abiertas
    - ▶ Implementaciones compatibles a priori

- ▶ **Inconvenientes:**

- ▶ HTTP es un protocolo simple y sin estado, por lo que **no dispone de servicios de apoyo.**

- ▶ Ej.: servicios de transacciones mejor en CORBA.

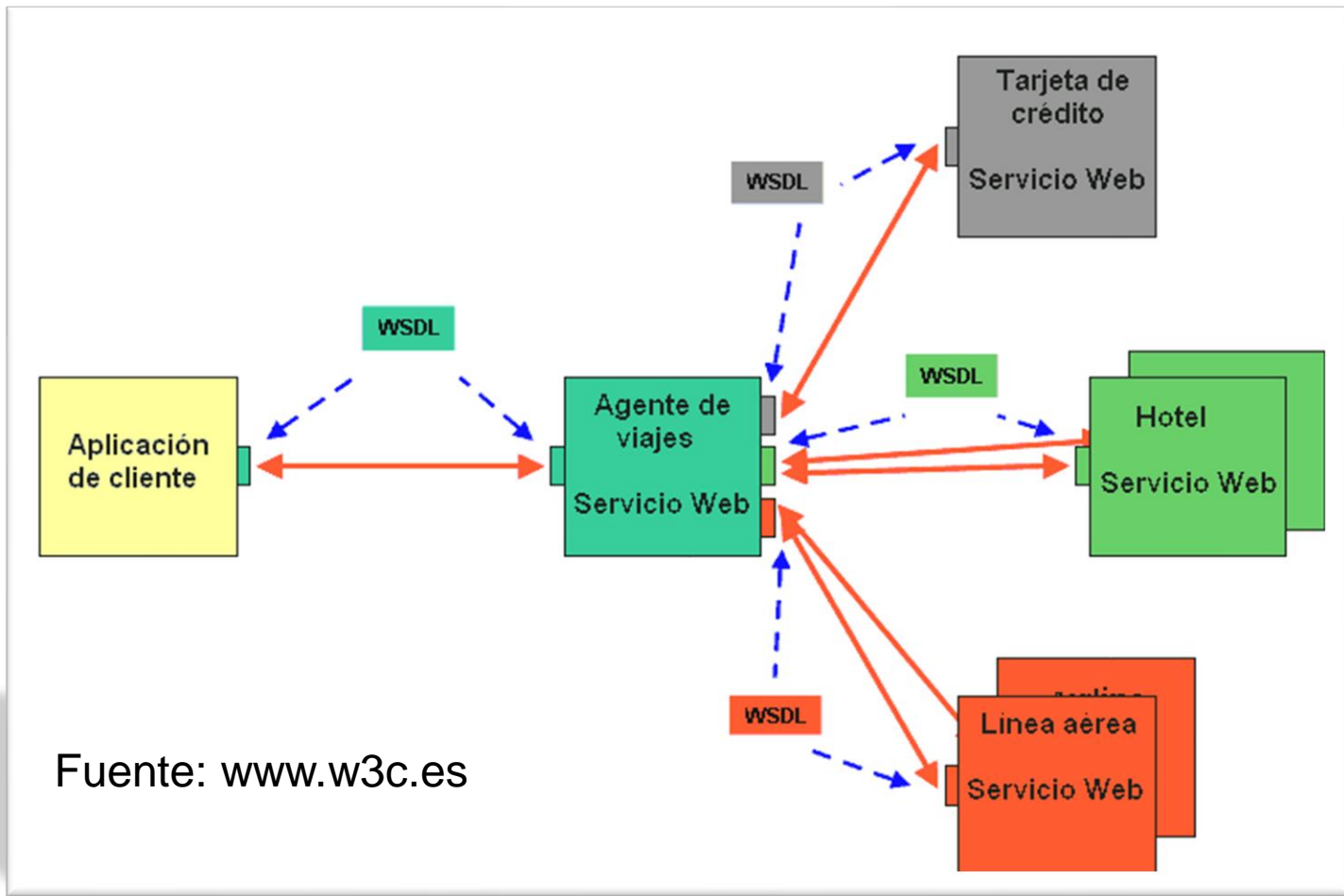
- ▶ **Rendimiento es más bajo** que otras soluciones.

- ▶ Ej.: mandar datos binarios comparado con RMI, CORBA o DCOM.
    - ▶ Preciso conversión a XML, lo que añade una mayor sobrecarga.

- ▶ **Potenciales problemas de seguridad.**

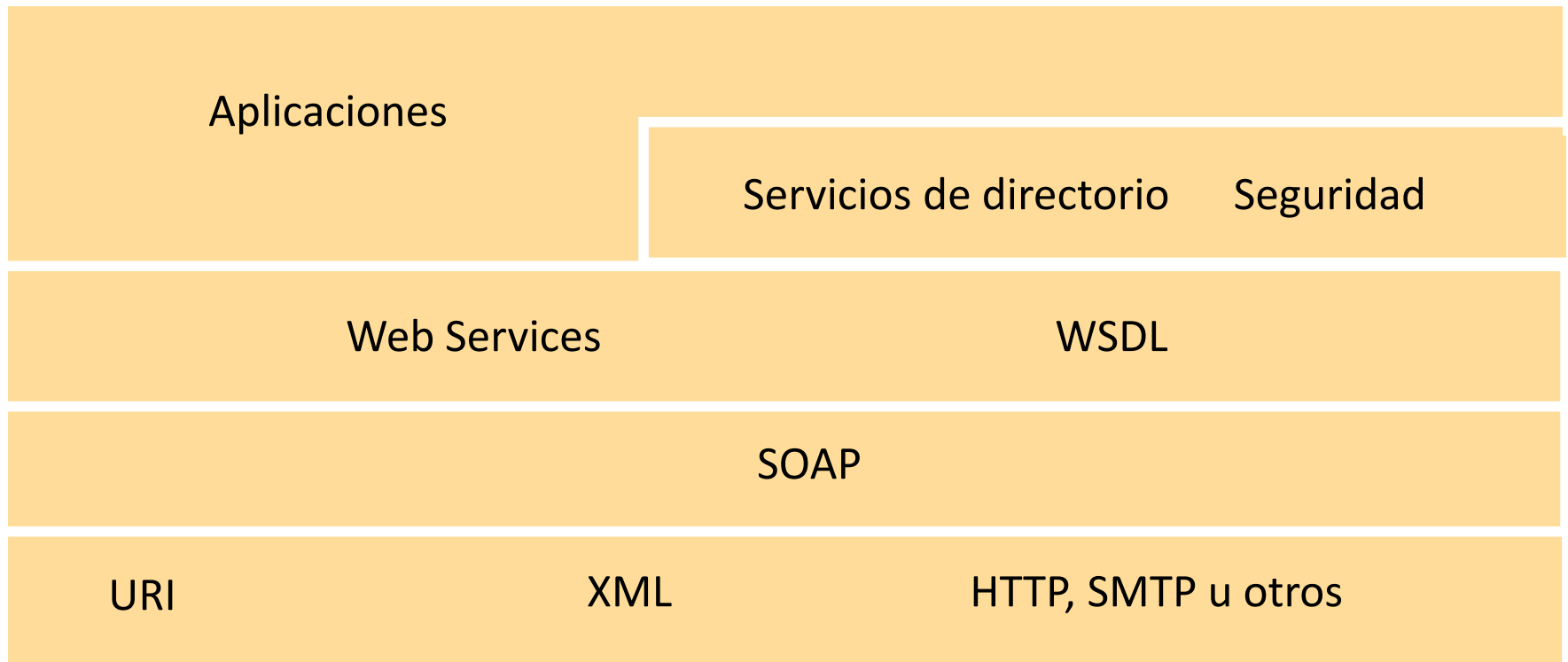
- ▶ Dado que los firewall dejan pasar el tráfico HTTP, puede ser preciso asegurar el acceso a los servicios.

# Combinación de servicios Web



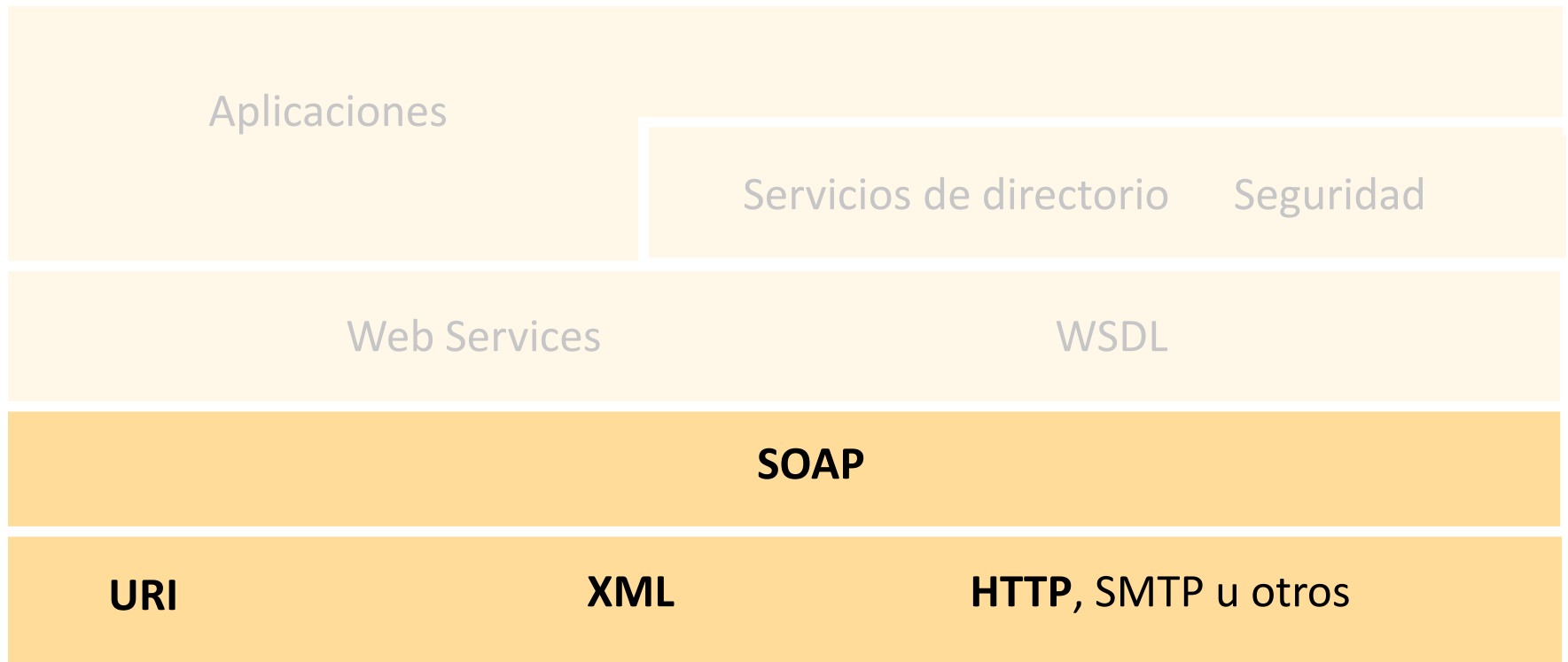
# Componentes e infraestructura

---



# Componentes e infraestructura

---



# Contenidos

---

## 1. Introducción:

1. Paradigma de servicios de red

## 2. **SOAP**

1. Introducción

2. **Arquitectura**

3. Ejemplo de aplicación

- ▶ Desarrollo de un servicio privado

# SOAP

---

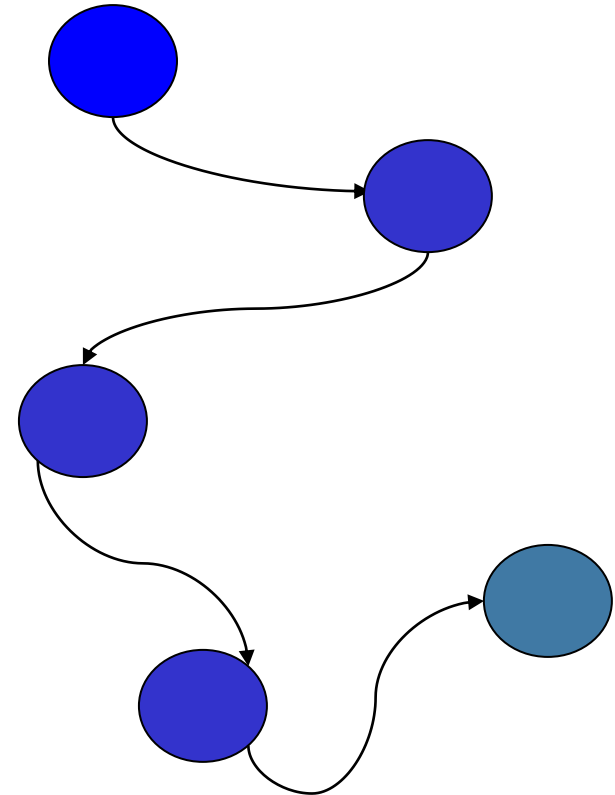
- ▶ *Simple Object Access Protocol*
  - ▶ <http://www.w3.org>
- ▶ **SOAP específica:**
  - ▶ Cómo representar los mensajes en XML
  - ▶ Como combinar mensajes SOAP para un modelo petición-respuesta
  - ▶ Cómo procesar los elementos de los mensajes
  - ▶ Cómo utilizar el transporte (HTTP, SMTP, ...)  
para enviar mensajes SOAP



# Nodo SOAP

---

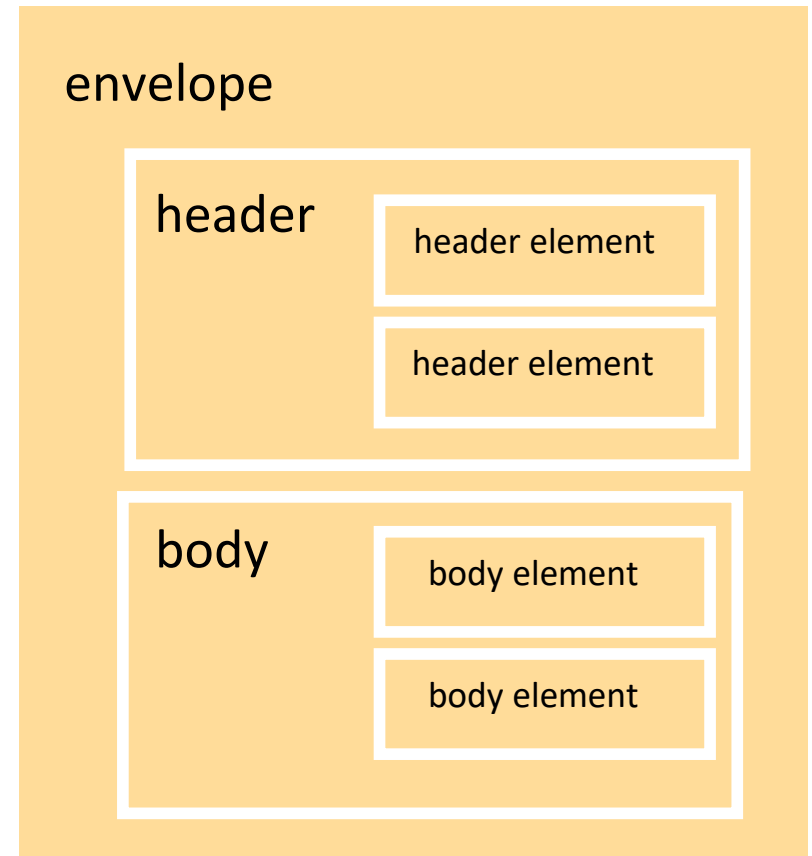
- ▶ **Nodo** que transmite, recibe, procesa y responde un **mensaje SOAP**
- ▶ Tipos de nodo:
  - ▶ Emisor SOAP
  - ▶ Receptor SOAP
  - ▶ Intermediario



# Mensaje SOAP

---

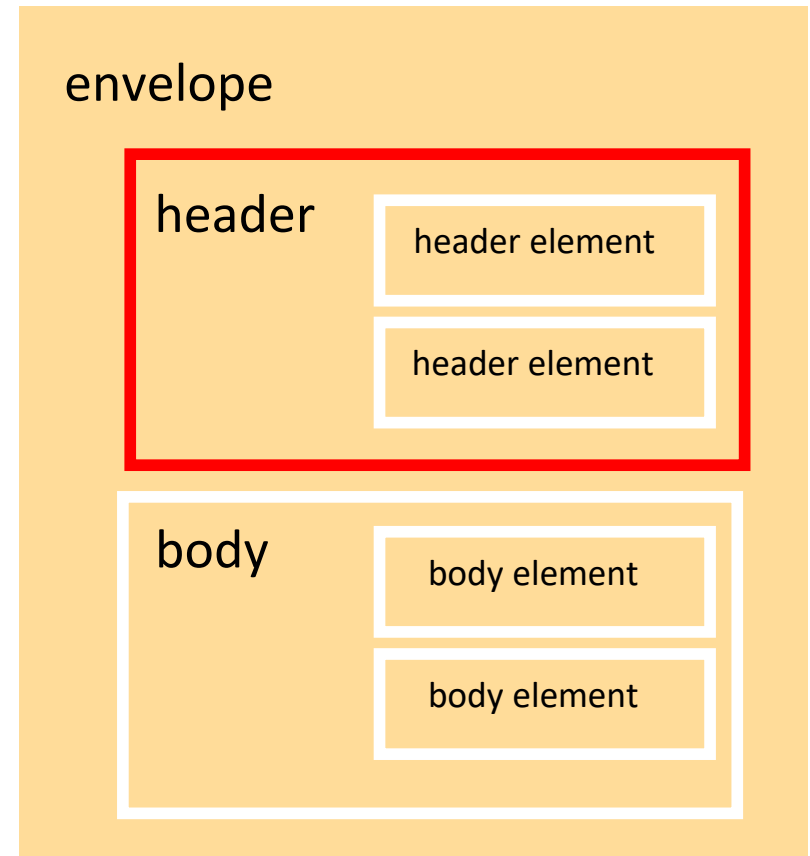
- ▶ Unidad básica de comunicación entre nodos SOAP
- ▶ El mensaje es transportado en un *envelope*
  - ▶ Encabezado opcional
  - ▶ Cuerpo
- ▶ Los elementos XML anteriores son definidos como un esquema en el espacio de nombres XML
  - ▶ Esquema definido en <http://www.w3.org>



# Mensaje SOAP: encabezado

---

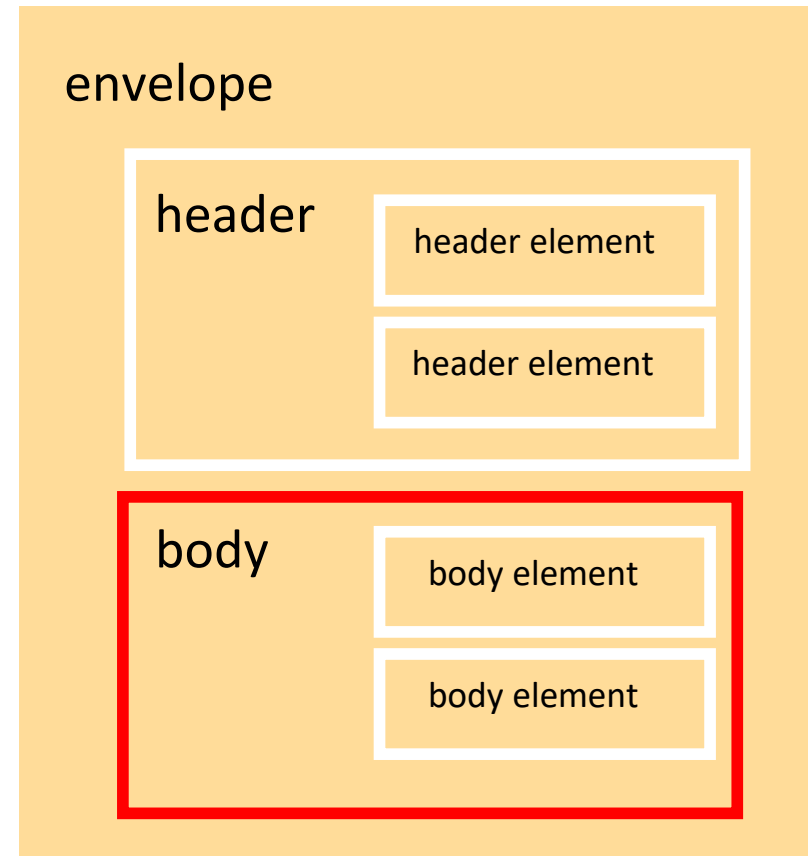
- ▶ Elemento **opcional**
- ▶ Incluye **información de control**:
  - ▶ **Identificador de transacción** para su uso con un servicio de transacciones
  - ▶ Un **identificador** de mensajes para **relacionar mensajes** entre sí
    - ▶ Los servicios son autónomos e independientes entre sí
  - ▶ Un **nombre de usuario**, una **clave pública**, etc.



# Mensaje SOAP: cuerpo

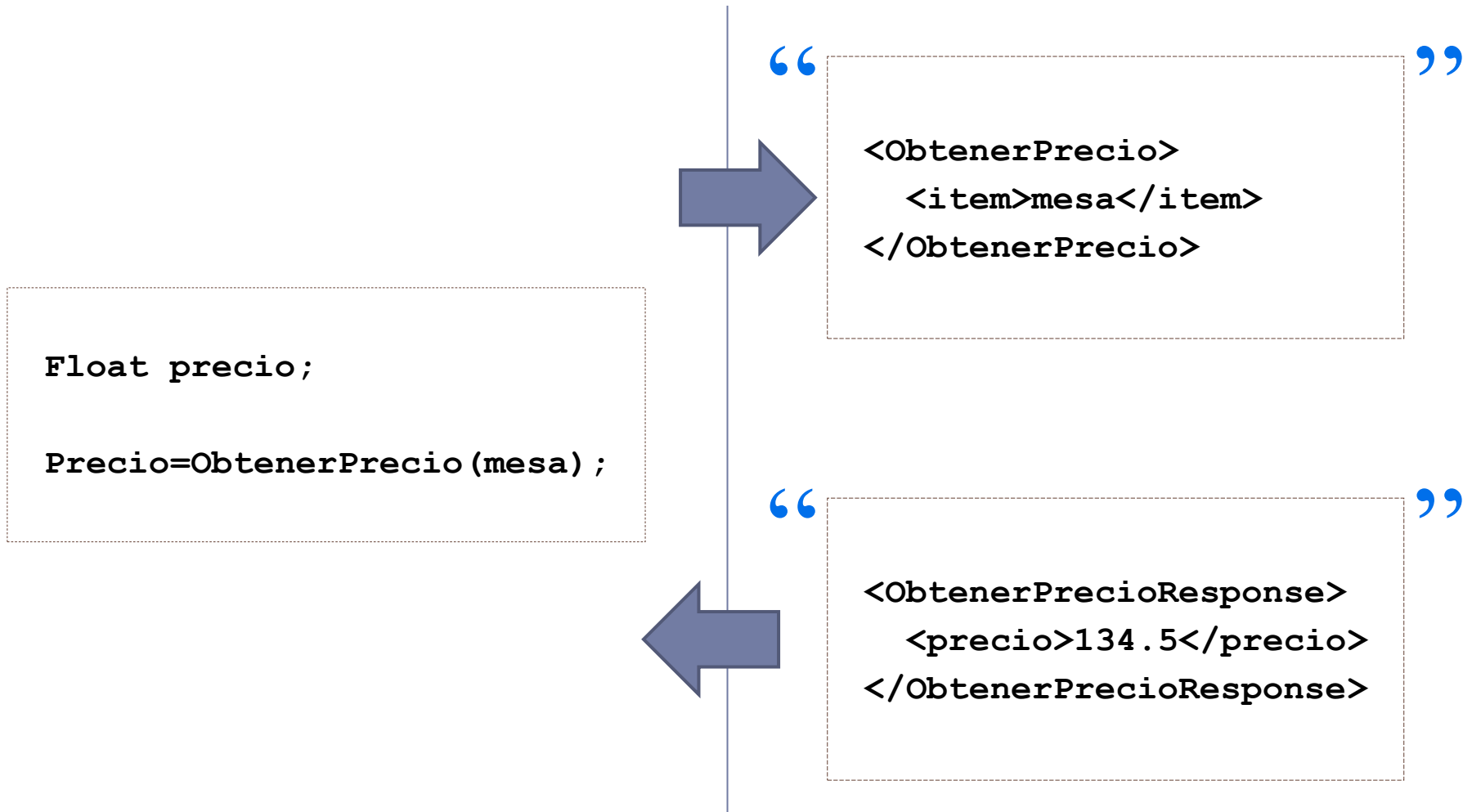
---

- ▶ Incluye la información:
  - ▶ Mensaje
  - ▶ Referencia al esquema XML que describe el servicio
- ▶ En los mensajes de una comunicación cliente/servidor (RPC):
  - ▶ El elemento *body* contiene una *petición* o una *respuesta*.



# Serialización en XML

---



# Transporte de mensajes SOAP

---

## ▶ Protocolo HTTP

### ▶ Estilo RPC:

- ▶ Petición: en HTTP POST
- ▶ Respuesta: en la respuesta al POST

### ▶ Envío de información:

- ▶ Con HTTP POST
- ▶ Con HTTP GET

## ▶ Protocolo SMTP

- ▶ La especificación indica cómo encapsular mensajes SOAP en mensajes con el formato usado en SMTP
  - ▶ Ejemplo: grandes volúmenes de datos binarios

# Ejemplo de petición/respuesta

---

“  
<ObtenerPrecio>  
    <item>mesa</item>  
</ObtenerPrecio>  
”



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:ObtenerPrecio xmlns:m="http://example.com/stockquote.xsd">  
      <item>mesa</item>  
    </m:ObtenerPrecio>  
  </SOAP-ENV:Body>  
</SOAP-ENV:Envelope>

# Ejemplo de petición/*respuesta*

---

“  
<ObtenerPrecioResponse>  
  <precio>134.5</precio>  
</ObtenerPrecioResponse>  
”



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:ObtenerPrecioResponse xmlns:m="http://example.com/stockquote.xsd">
      <Precio>134.5</Precio>
    </m:ObtenerPrecioResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



# Contenidos

---

## 1. Introducción:

1. Paradigma de servicios de red

## 2. **SOAP**

1. Introducción

2. Arquitectura

### 3. **Ejemplo de aplicación**

- ▶ **Desarrollo de un servicio privado**

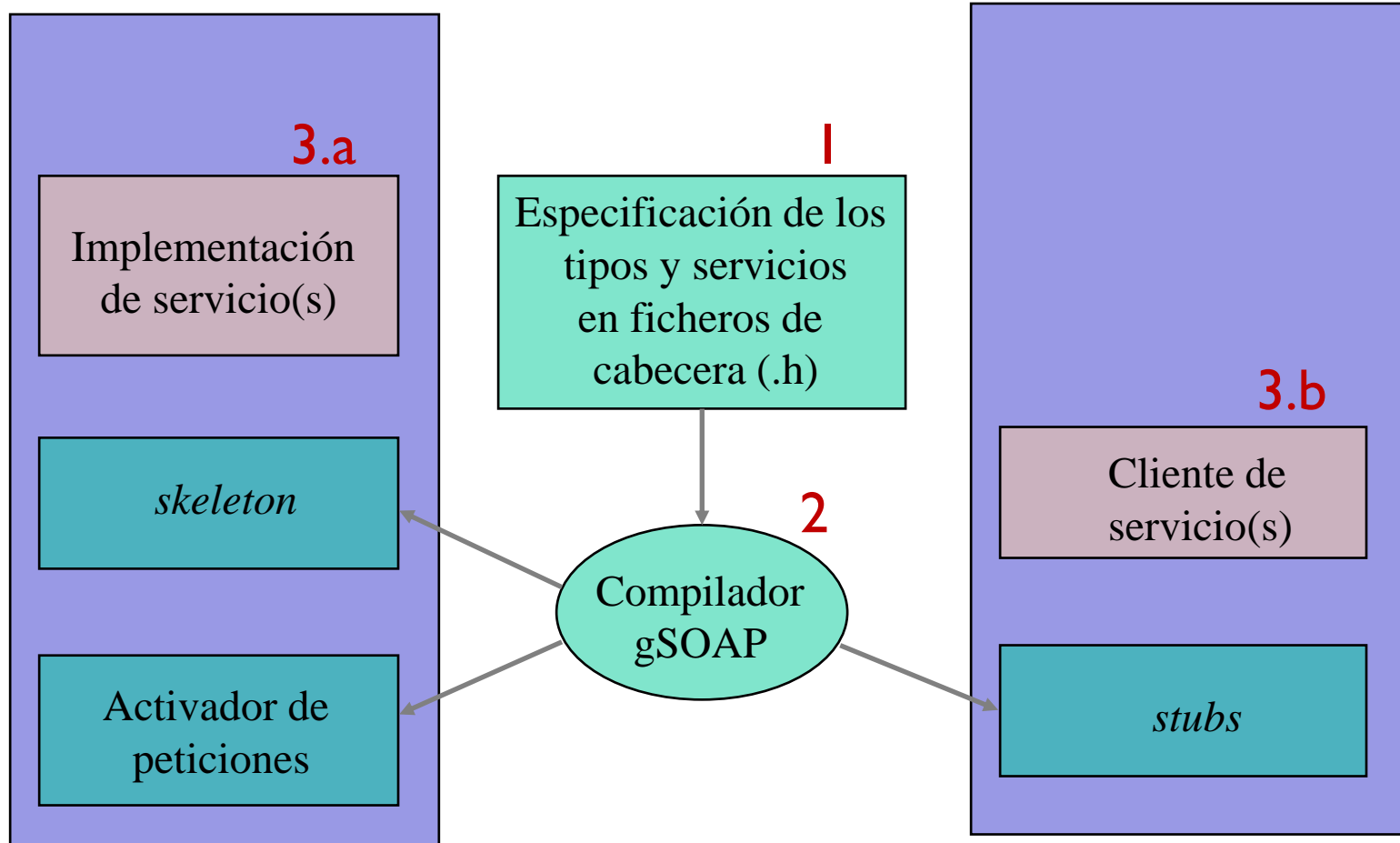
# Plataforma de desarrollo

---

- ▶ **gSOAP**

- ▶ Conjunto de herramientas para el desarrollo de aplicaciones basadas en servicios Web en C/C++
- ▶ <http://www.cs.fsu.edu/~engelen/soap.html>

# Desarrollo de un servicio privado



# calc.h

---

- Descripción de las funciones que dan acceso a los servicios de la interfaz
- Uso de lenguaje C o Java.

```
int ns__suma (int a, int b, int *res);  
int ns__resta (int a, int b, int *res);
```

# Preprocesado de la interfaz a C (1/2)

---

```
acaldero@guernika# soapcpp2 -c calc.h
```

- ▶ Genera los siguientes archivos C en el directorio:
  - ▶ `soapC.c`: Serialización de las operaciones
  - ▶ `soapClient.c`: Resguardo (*stub*) del cliente
  - ▶ `soapClientLib.c`: Incluye código necesario del lado del cliente
  - ▶ `soapH.h`: Interfaz de serialización
  - ▶ `soapServer.c`: Esqueleto (*skeleton*) del servidor
  - ▶ `soapServerLib.c`: Incluye código necesario del lado del servidor
  - ▶ `soapStub.h`: Interfaz del resguardo y del esqueleto
  - ▶ `ns.nsmmap`: Identificación del espacio de nombre (entorno)

# Preprocesado de la interfaz a C (2/2)

---

```
acaldero@guernika# soapcpp2 -c calc.h
```

- ▶ Genera los siguientes archivos XML en el directorio:
  - ▶ `ns.resta.req.xml`: Descripción argumentos entrada a resta
  - ▶ `ns.resta.res.xml`: Descripción valor de retorno de resta
  - ▶ `ns.suma.req.xml`: Descripción argumentos entrada a resta
  - ▶ `ns.suma.res.xml`: Descripción valor retorno de resta
  - ▶ `ns.wsdl`: Descripción como servicio Web
  - ▶ `ns.xsd`: Descripción de las operaciones de la interfaz

# Preprocesado de la interfaz a C

---

```
acaldero@guernika# soapcpp2 -c calc.h
```

- ▶ NO genera (y el programador ha de escribir):
  - ▶ `calcServer.c`: Servidor SOAP e implementación de interfaz.
  - ▶ `calcClient.c`: Ejemplo de cliente SOAP.

# calcServer.c (1 / 3)

---

```
#include "soapH.h"
#include "ns.nsmap"

int main(int argc, char **argv)
{
    int m, s; /* sockets del cliente (s) y servidor (m) */
    struct soap soap;

    if (argc < 2)
    {
        printf("Usage: %s <port>\n", argv[0]); exit(-1);
    }

    soap_init(&soap);
```



# calcServer.c (2/3)

```
m = soap_bind(&soap, NULL, atoi(argv[1]), 100);
if (m < 0) {
    soap_print_fault(&soap, stderr); exit(-1);
}

while (1) {
    s = soap_accept(&soap);
    if (s < 0) {
        soap_print_fault(&soap, stderr); exit(-1);
    }
    soap_serve(&soap);
    soap_end(&soap);
}

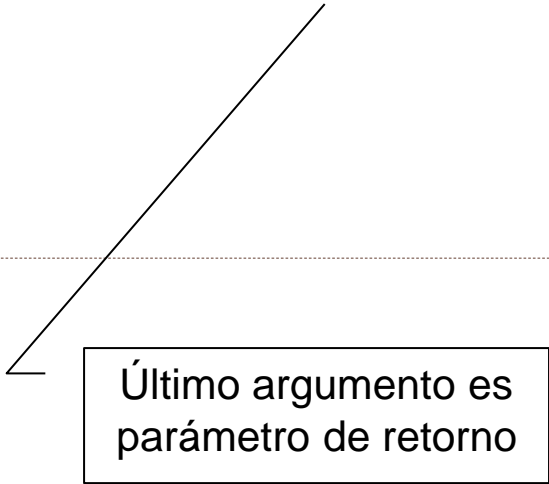
return 0;
}
```

# calcServer.c (3/3)

---

```
int ns__suma (struct soap *soap, int a, int b, int *res)
{
    *res = a + b;
    return SOAP_OK;
}
```

```
int ns__resta (struct soap *soap, int a, int b, int *res)
{
    *res = a - b;
    return SOAP_OK;
}
```



Último argumento es  
parámetro de retorno

# calcClient.c (1 / 2)

---

```
#include "soapH.h"
#include "ns.nsmapi"

int main(int argc, char **argv)
{
    struct soap soap;
    char *serverURL;
    int a, b, res;

    if (argc != 4) {
        printf("Uso: %s http://servidor:puerto numero1 numero2\n", argv[0]);
        exit(0);
    }

    soap_init(&soap);
```

# calcClient.c (2/2)

---

```
serverURL = argv[1];
a = atoi(argv[2]) ;
b = atoi(argv[3]) ;

soap_call_ns__suma(&soap, serverURL, "", a, b, &res);
if (soap.error) {
    soap_print_fault(&soap, stderr); exit(1);
}

printf("Resultado = %d \n", res);

soap_destroy(&soap);
soap_end(&soap);
soap_done(&soap);

return 0;
}
```

# Despliegue del ejemplo

guernika.lab.inf.uc3m.es

---

```
acaldero@guernika # ls -w 40
```

```
calcClient.c
```

```
calc.h
```

```
calcServer.c
```

```
ns.nsmmap
```

```
ns.resta.req.xml
```

```
ns.resta.res.xml
```

```
ns.suma.req.xml
```

```
ns.suma.res.xml
```

```
ns.wsdl
```

```
ns.xsd
```

```
soapC.cpp
```

```
soapClient.cpp
```

```
soapClientLib.cpp
```

```
soapH.h
```

```
soapObject.h
```

```
soapProxy.h
```

```
soapServer.cpp
```

```
soapServerLib.cpp
```

```
soapStub.h
```

# Compilación del ejemplo

guernika.lab.inf.uc3m.es

---

```
# gcc -Wall -g -I/opt/gsoap-linux-2.7/ -c soapC.c -o soapC.o
# gcc -Wall -g -I/opt/gsoap-linux-2.7/ -c calcClient.c -o calcClient.o
# gcc -Wall -g -I/opt/gsoap-linux-2.7/ -c soapClient.c -o soapClient.o
# gcc -Wall -g -I/opt/gsoap-linux-2.7/ -c calcServer.c -o calcServer.o
# gcc -Wall -g -I/opt/gsoap-linux-2.7/ -c soapServer.c -o soapServer.o
# gcc -Wall -g -I/opt/gsoap-linux-2.7/ \
    -c /opt/gsoap-linux-2.7/stdsoap2.c -o stdsoap2.o

# gcc -o client calcClient.o soapC.o soapClient.o stdsoap2.o
# gcc -o server calcServer.o soapC.o soapServer.o stdsoap2.o
```

# Ejecución del ejemplo

guernika.lab.inf.uc3m.es

---

```
acaldero@guernika # ./server 9000
```

```
acaldero@guernika # ./client http://localhost:9000 10 12
```

# Aplicaciones de Internet: SOAP

Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid