

Grupo ARCOS



uc3m | Universidad **Carlos III** de Madrid

Tema 8: Servicios Web

Sistemas Distribuidos



Grado en Ingeniería Informática

Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. XML, SOAP, WSDL, UDDI

3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

Contenidos

1. Introducción:

1. **Paradigma de servicios de red**
2. Servidor Web también para llamadas remotas

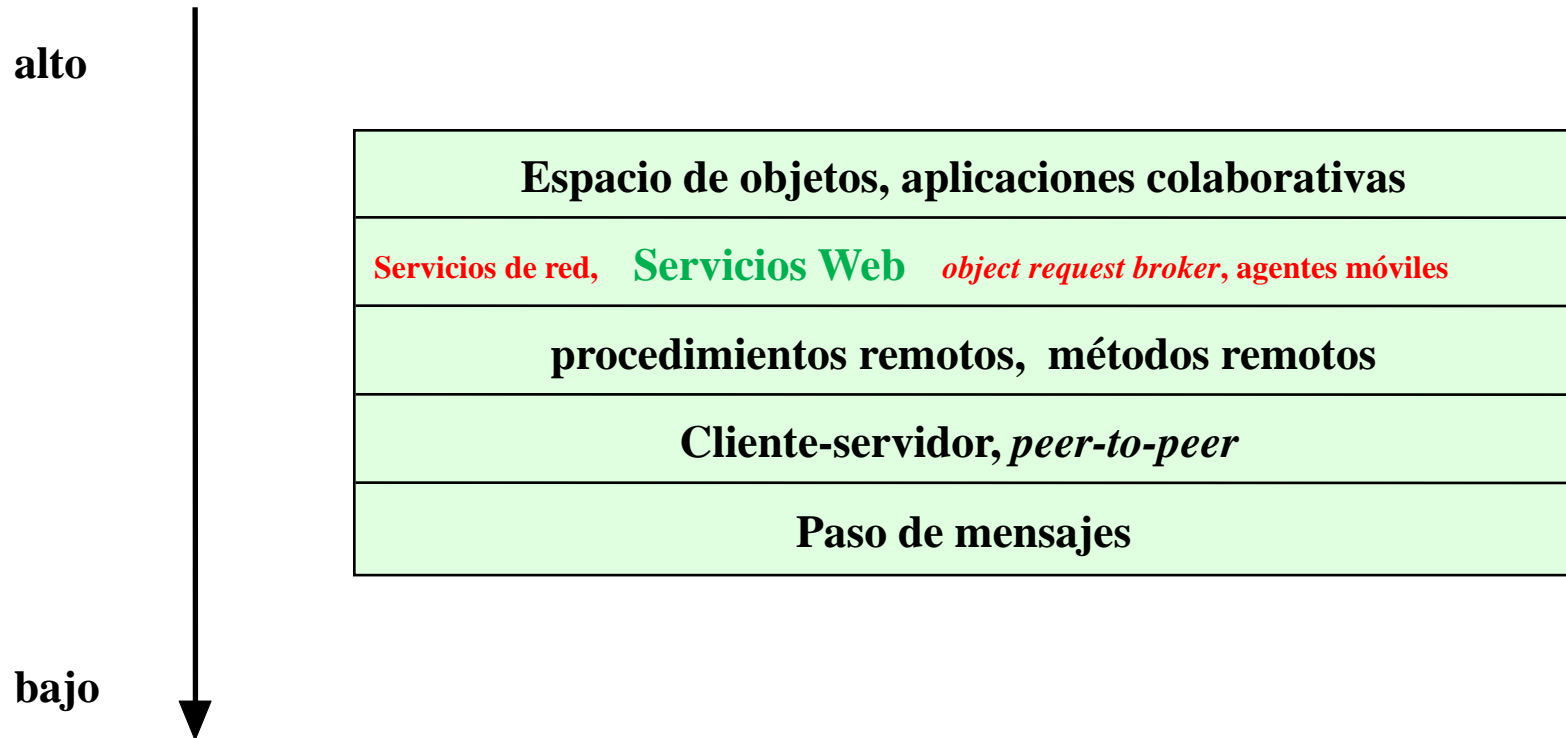
2. Elementos en un Servicio Web:

1. XML, SOAP, WSDL, UDDI

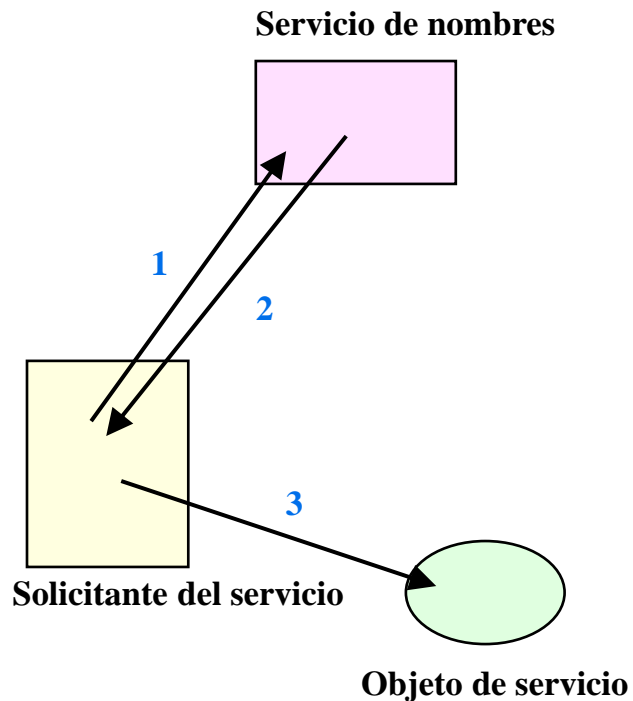
3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (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 nombres**
 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. **Servidor Web también para llamadas remotas**

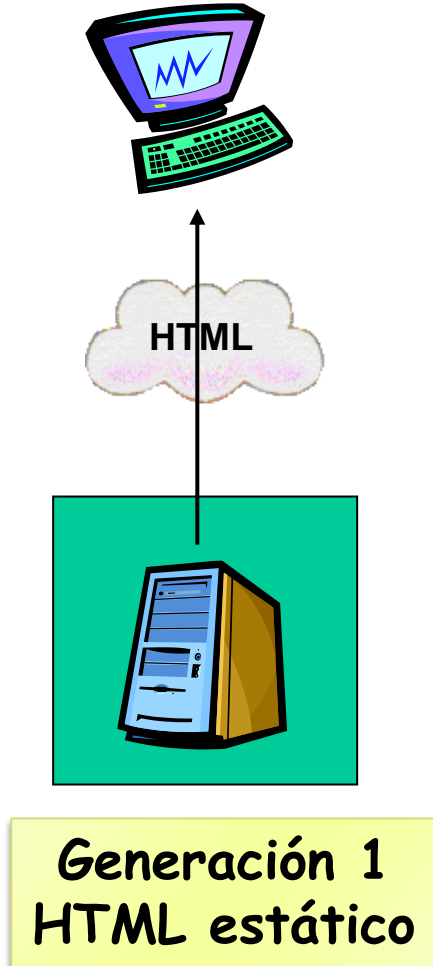
2. Elementos en un Servicio Web:

1. XML, SOAP, WSDL, UDDI

3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

Evolución de la Web...

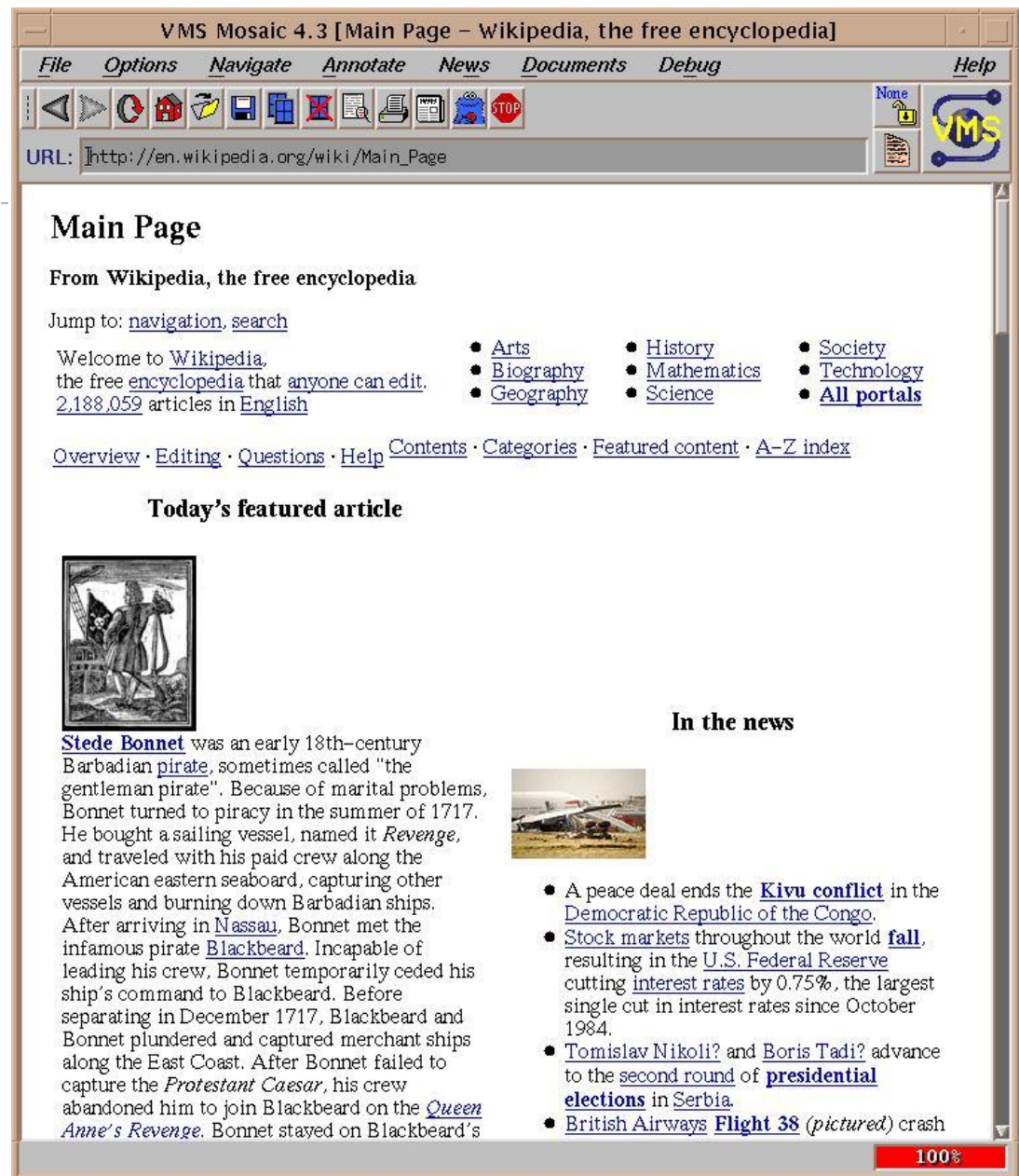
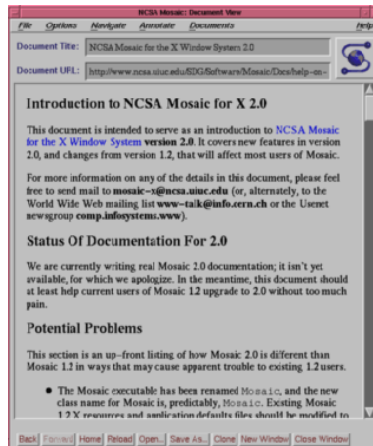
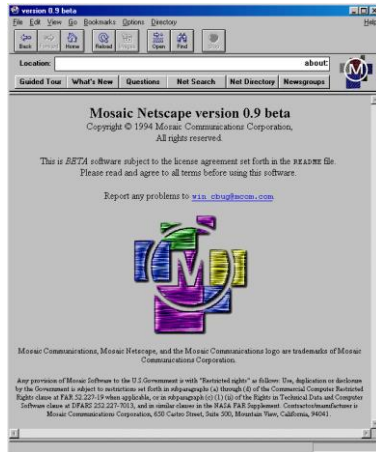


- ▶ El navegador Web pide una página Web indicando su identificador URI en la petición.
- ▶ El servidor Web busca el fichero almacenado que se corresponde con la URI pedida, y lo envía como respuesta.
- ▶ Se utiliza el protocolo HTTP para la transferencia de contenido.
- ▶ Contenido diverso:
 - ▶ Páginas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

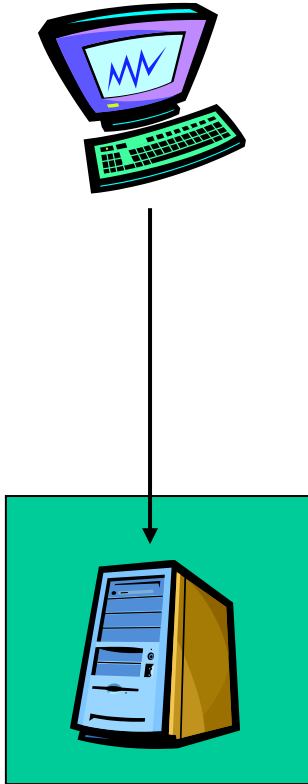
Ejemplo de la generación 1



Ejemplo de la generación 1



Ejemplo de la generación 1: HTTP 1.0

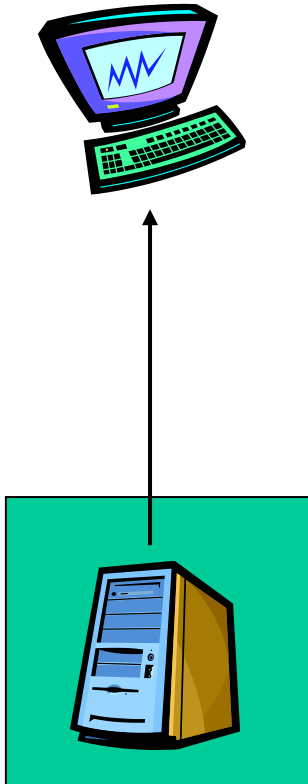


```
telnet www.uc3m.es 80
```

```
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>

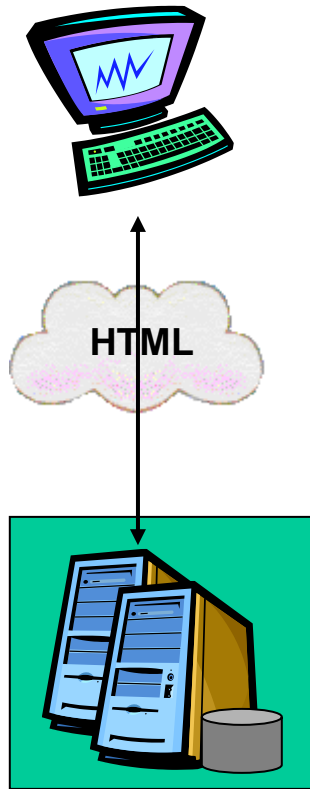
</head>

<body>

Hello world!

</BODY></HTML>

Evolución de la Web...



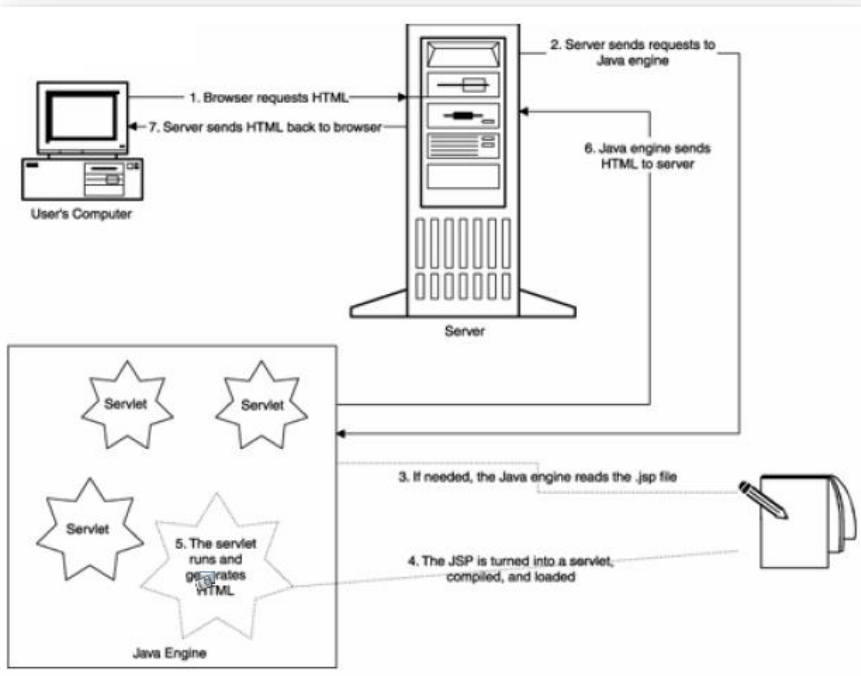
Generación 2 Aplicaciones Web

- ▶ 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
 - Ej.: **CGI**, **servlets** de Java, **lenguajes embebidos** (PHP, **JSP**, **ASP**, etc.)
 - ▶ En el cliente:
 - ▶ Además de páginas, imágenes, videos, etc. transferencia de aplicaciones para su **ejecución en el cliente** (navegador Web)
 - Ej.: **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 está 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);
    }
}
```

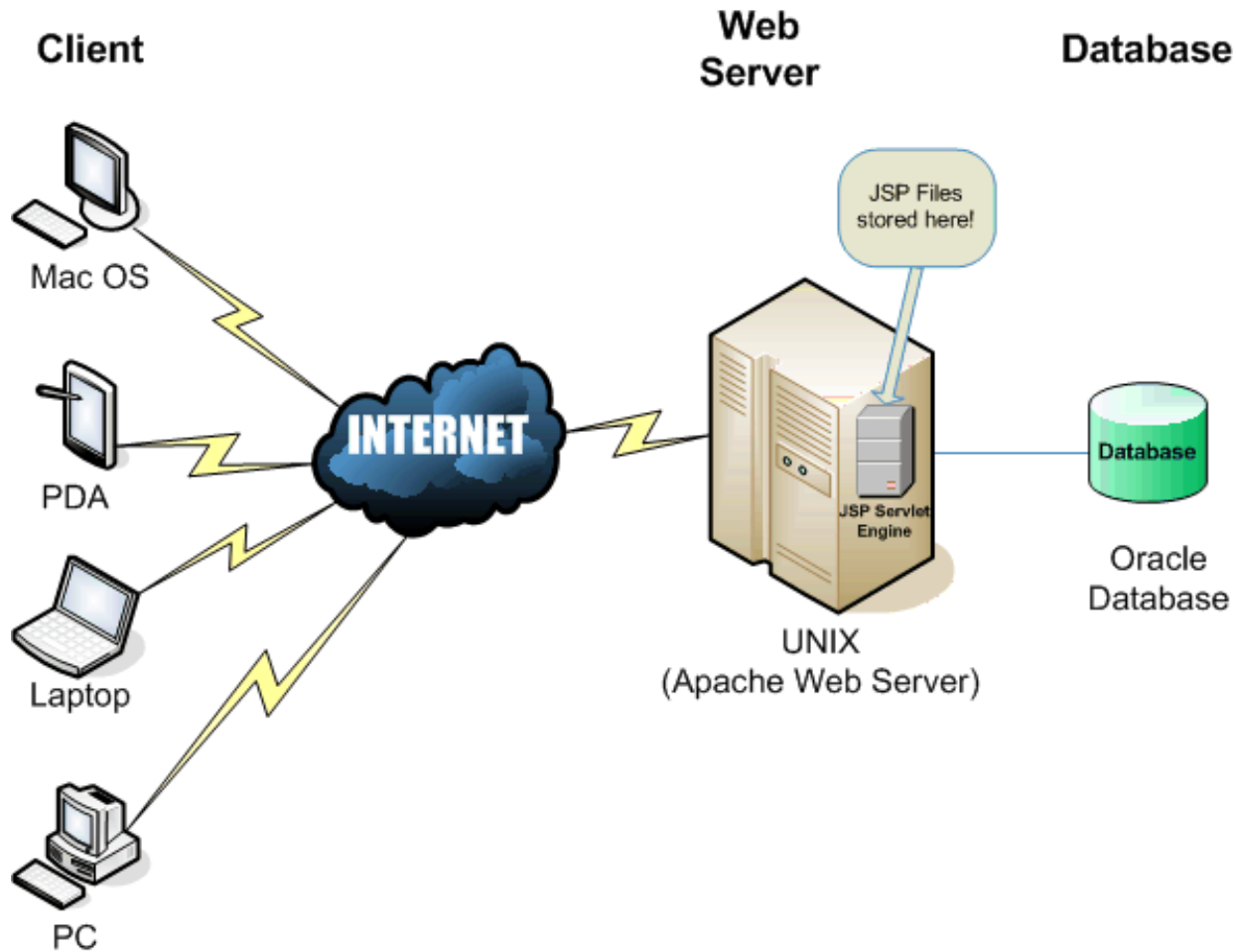
Ejemplo de la generación 2: JSP

- ▶ 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

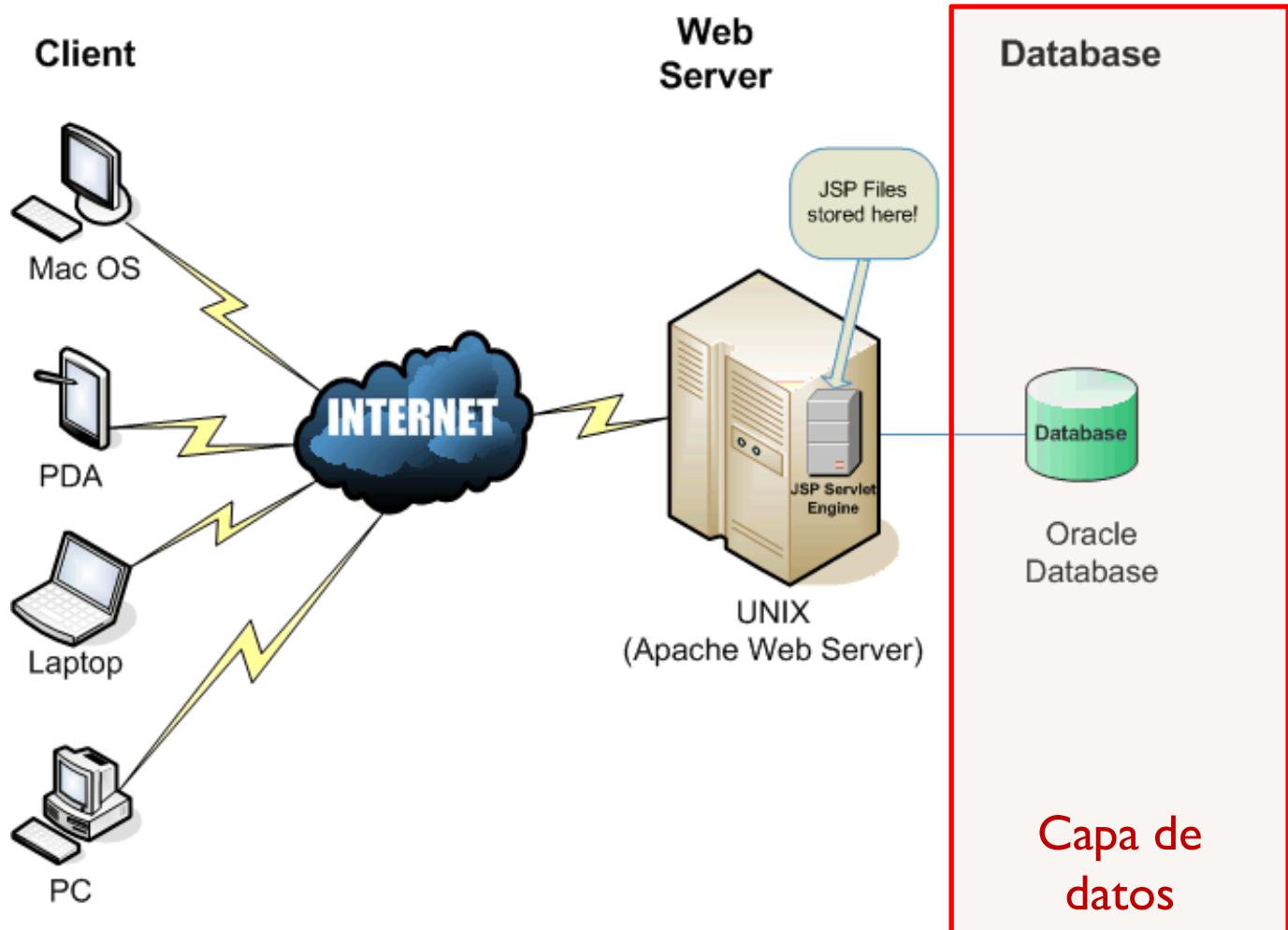
Ejemplo de la generación 2: *JSP*

```
<%@ 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>
```

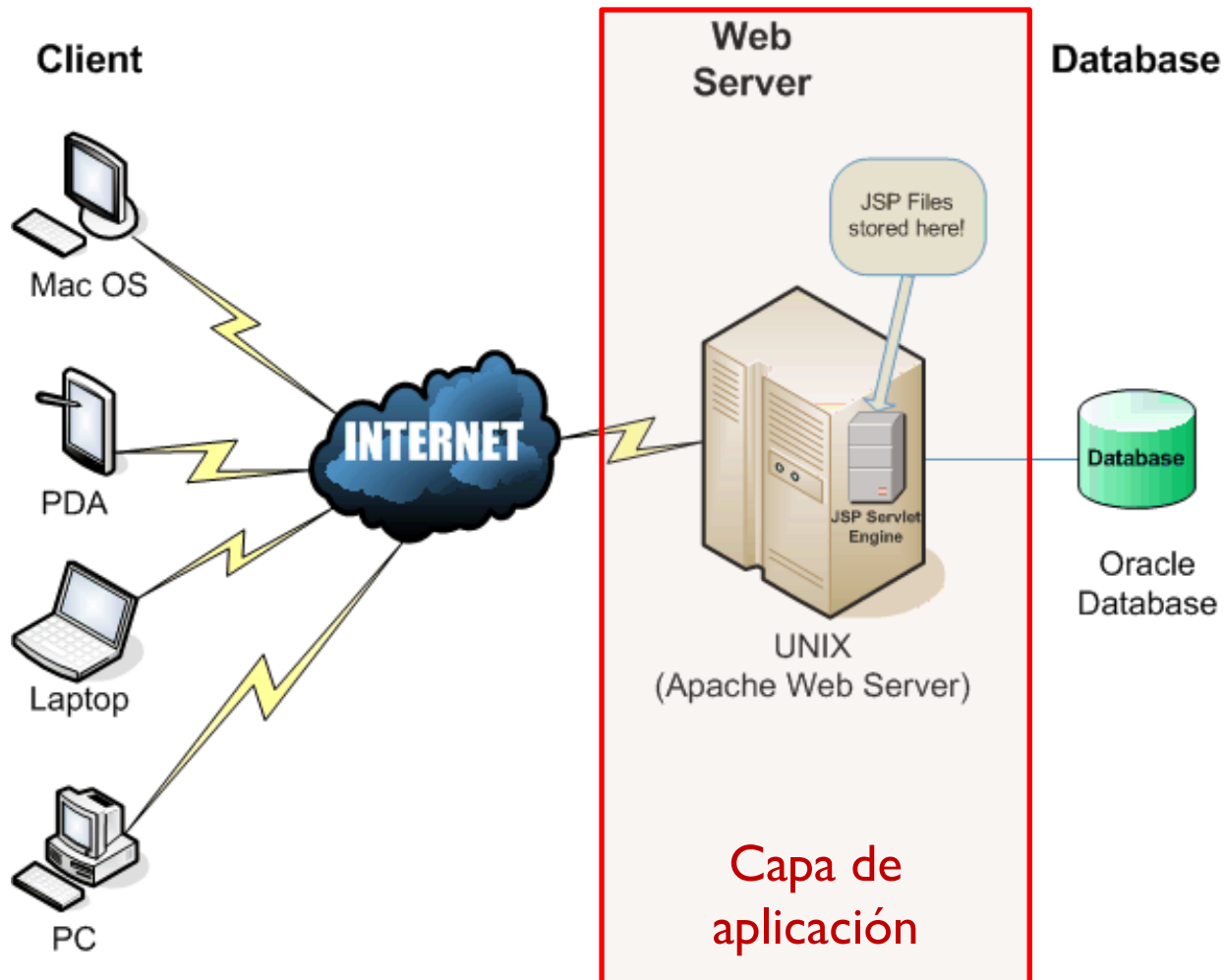
Arquitectura en tres capas (3-tier)



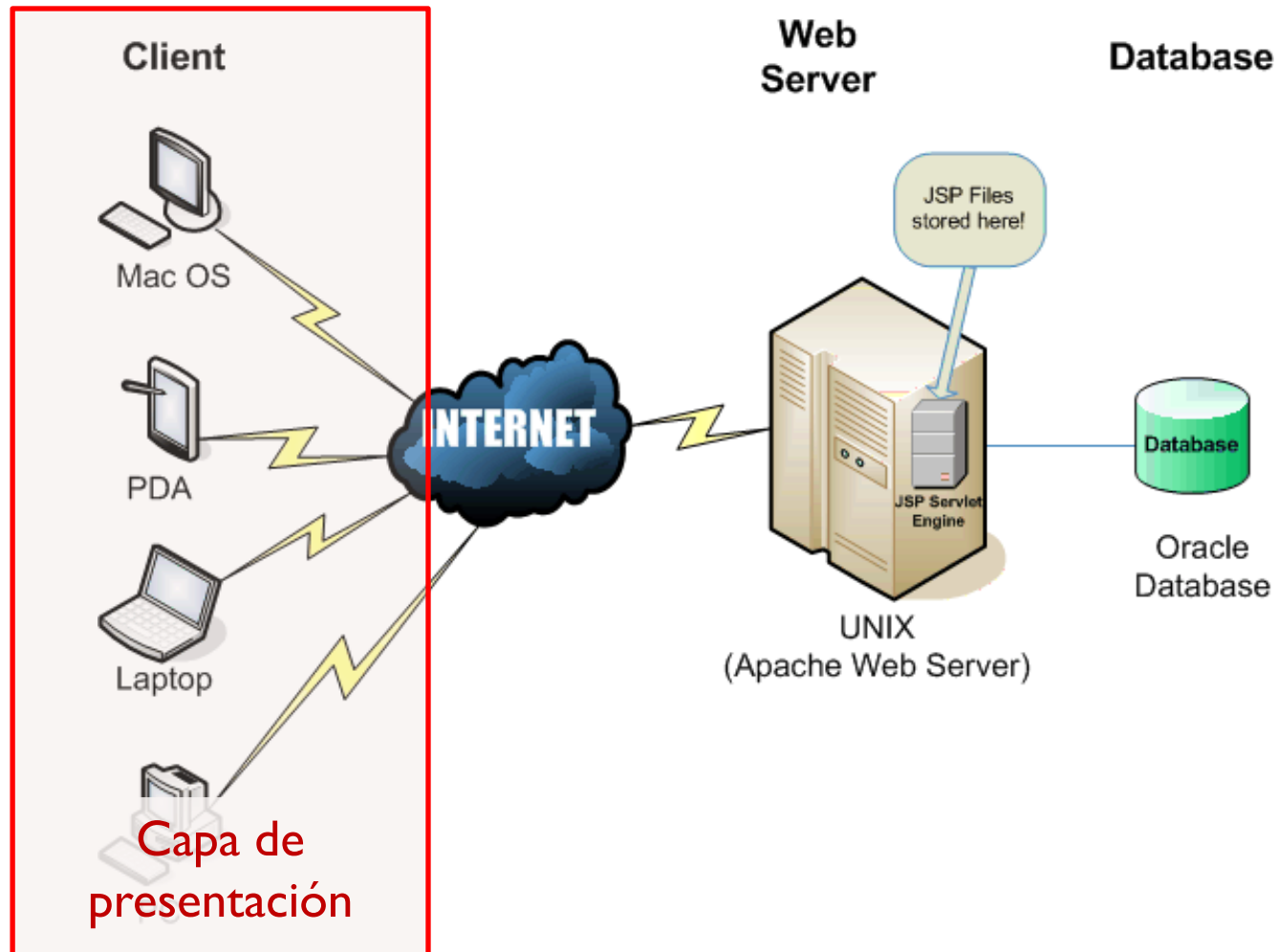
Arquitectura en tres capas (3-tier)



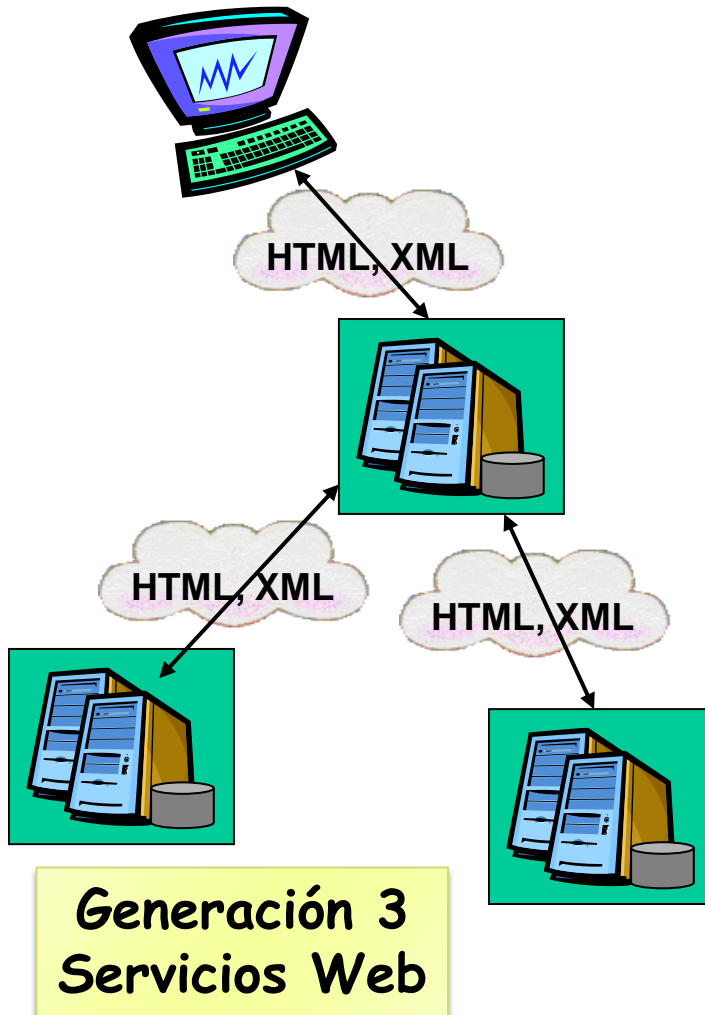
Arquitectura en tres capas (3-tier)



Arquitectura en tres capas (3-tier)

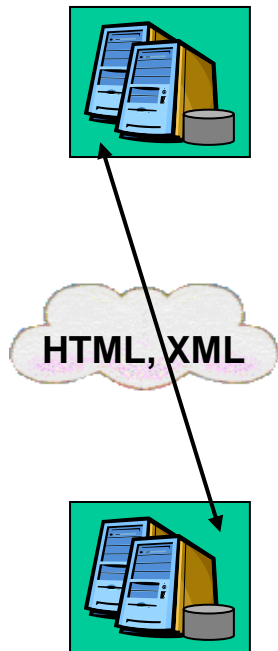


Evolución de la Web...



- ▶ 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.

SOA: *Service Oriented Architecture*

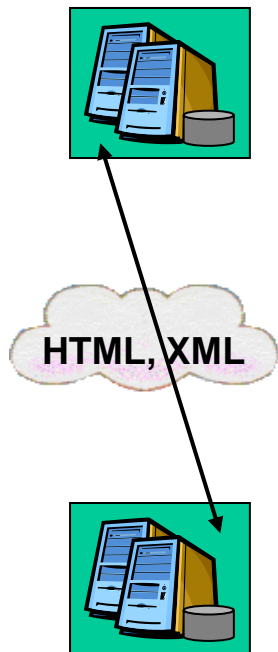
► Arquitectura SOA

- **Arquitectura en la que el software se expone como “servicio”, que es invocado utilizando un protocolo estándar de comunicación**
- Permite que diferentes aplicaciones puedan interoperar entre ellas
- **Las aplicaciones se componen de servicios modulares independientes que pueden interoperar**
- Permite el desarrollo de arquitecturas débilmente acopadas
- Suele utilizar un modelo basado en el paradigma cliente-servidor

► Implementaciones de SOA

- **Los servicios web se han convertido en la implementación más utilizada en arquitecturas orientadas a servicios**
- Estilos de servicios web
 - Servicios web SOAP
 - REST (RESTful Architecture Style)

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.

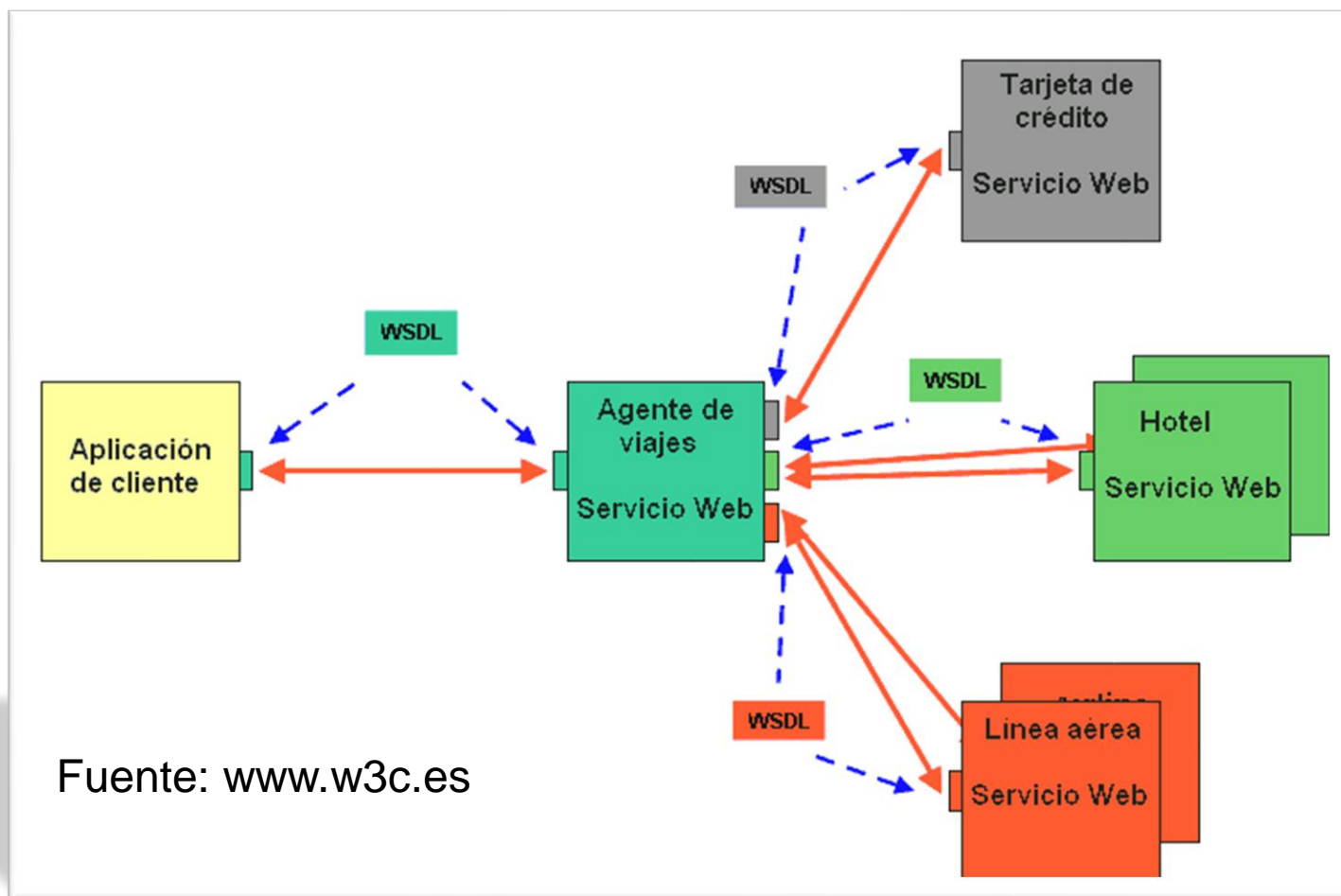
▶ Rendimiento es más bajo que otras soluciones.

- ▶ 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



Servicio Web

► Principales protocolos usados:

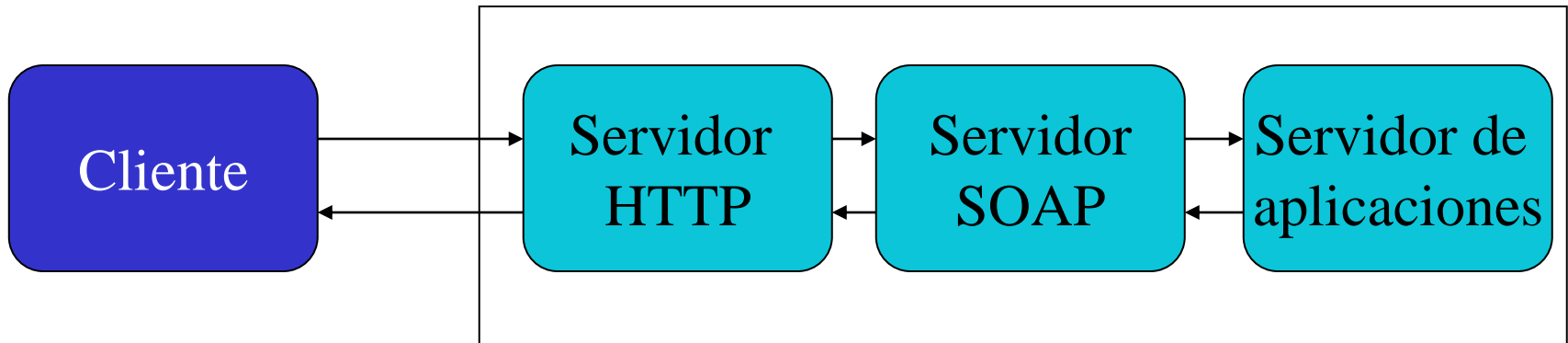


Aplicaciones		
	Servicios de directorio	Seguridad
Web Services		WSDL
SOAP		
URI	XML	HTTP, SMTP u otros

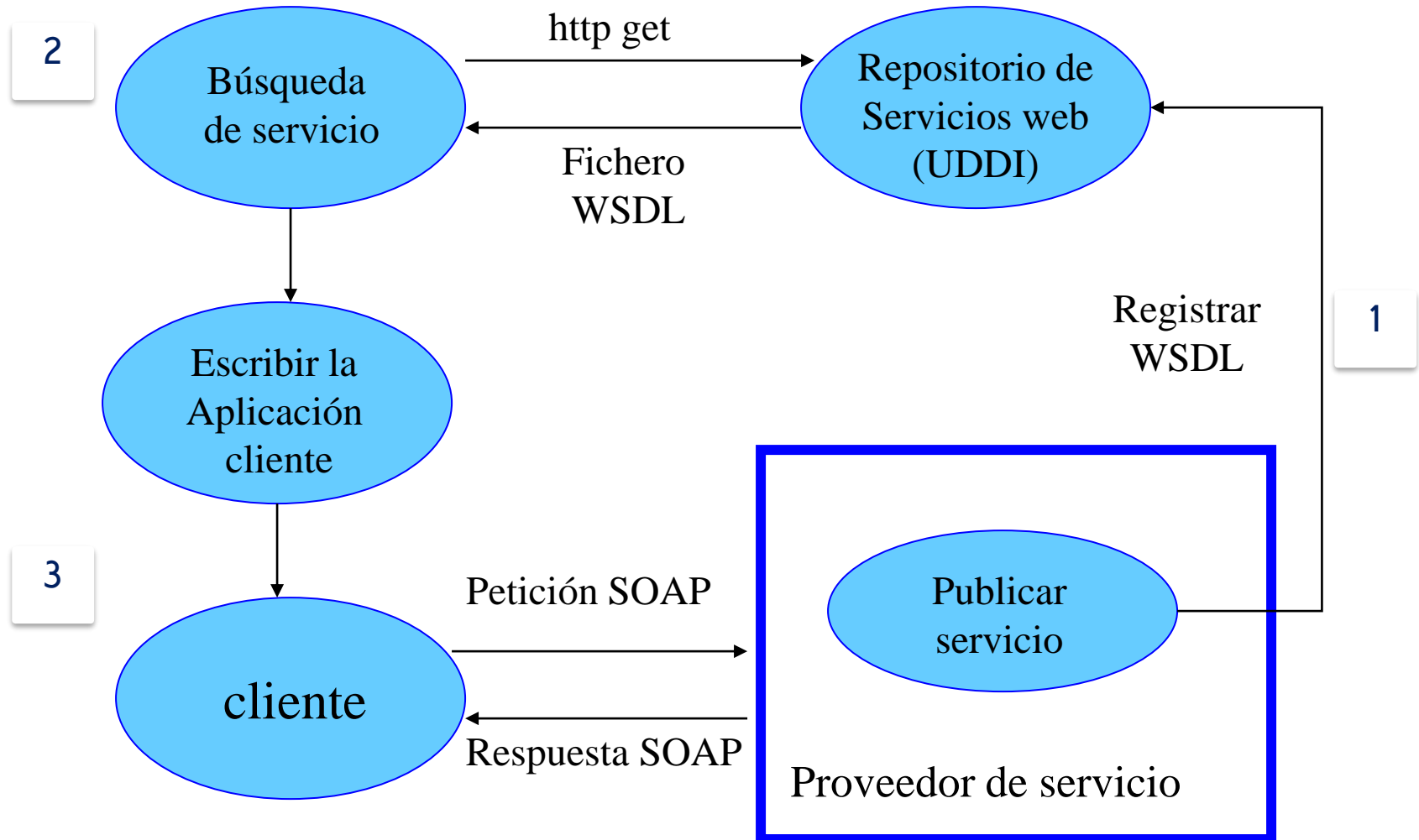
- HTTP: transporte utilizado
- XML: describe la información, los mensajes
- SOAP: empaqueta la información y la transmite entre el cliente y el proveedor del servicio
- WSDL: descripción del servicio
- UDDI: lista de servicios disponibles

Ejemplo de implantación

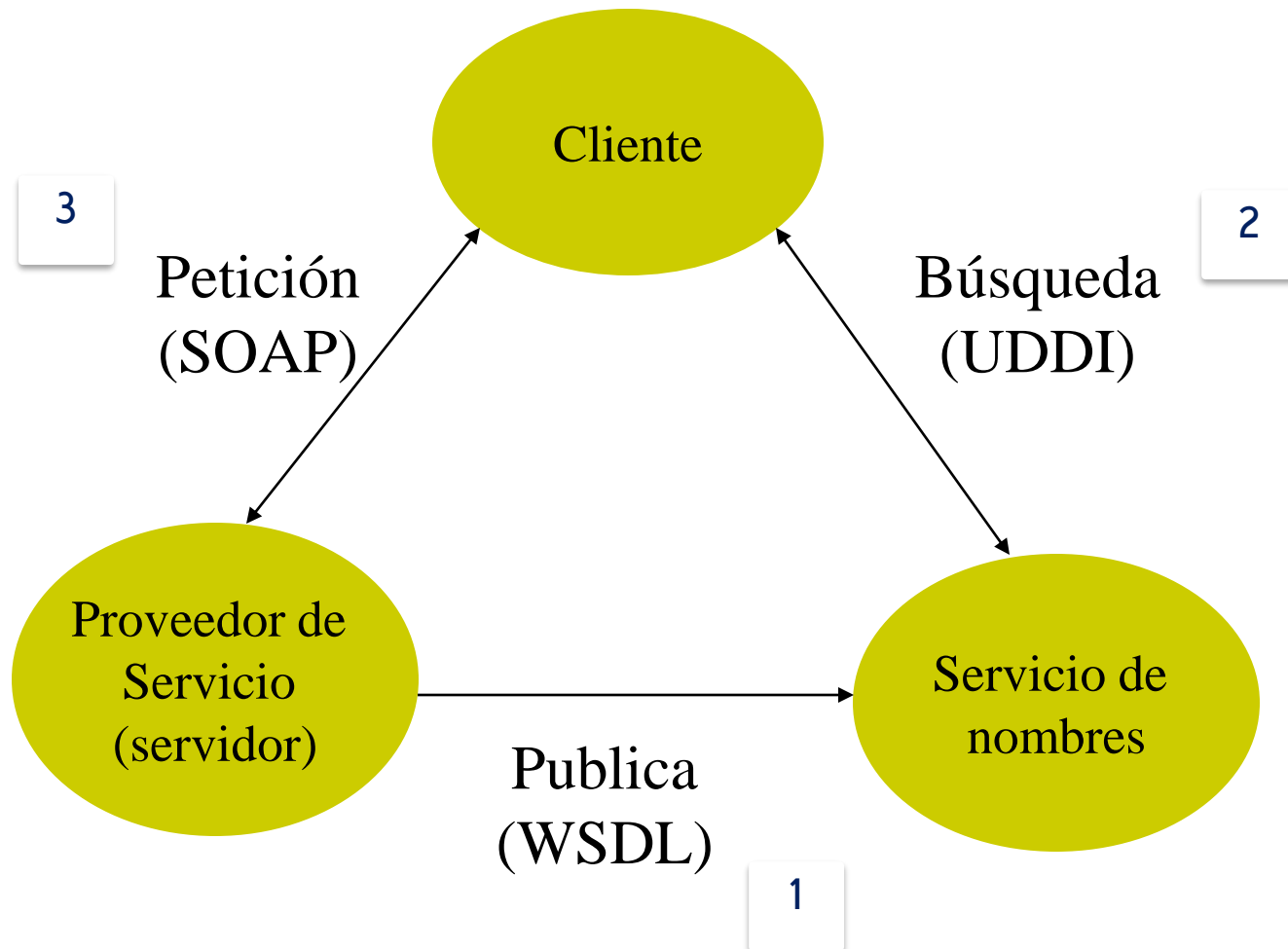
Proveedor del servicio Web



Escenario de uso



Servicios Web y SOA



Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. **XML**, SOAP, WSDL, UDDI

3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

XML

- *Extensible markup language*
 - Definido por W3C (<http://www.w3c.org>)
- XML es extensible, permite a los usuarios definir sus propias etiquetas (diferente a HTML)
- Componentes:
 - Elementos y atributos
 - `<tag attr=valor/>`
 - `<tag>valor</tag>`
 - Espacios de nombres
 - `xmlns="http://www.w3.org/1999/xhtmll"`
 - Esquemas
 - Elementos y atributos que pueden aparecer en un documento

Ejemplo de XML

- Ej: *float ObtenerPrecio(string item);*

Petición:

```
<ObtenerPrecio>  
  <item>mesa</item>  
</ObtenerPrecio>
```

Respuesta:

```
<ObtenerPrecioResponse>  
  <Precio>134.5</Precio>  
</ObtenerPrecioResponse>
```

Ejemplo de XML

- Ej: *float ObtenerPrecio(string item);*

Petición:

```
<ObtenerPrecio>  
  <item>mesa</item>  
</ObtenerPrecio>
```

Respuesta:

```
<ObtenerPrecioResponse>  
  <Precio>134.5</Precio>  
</ObtenerPrecioResponse>
```

Esquema:

```
<element name="ObtenerPrecio">  
  <complexType><all>  
    <element name="item" type="string"/>  
  </all></complexType>  
</element>  
<element name="ObtenerPrecioResponse">  
  <complexType><all>  
    <element name="Precio" type="float"/>  
  </all></complexType>  
</element>
```

Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. XML, **SOAP**, WSDL, UDDI

3. Ejemplo de aplicación

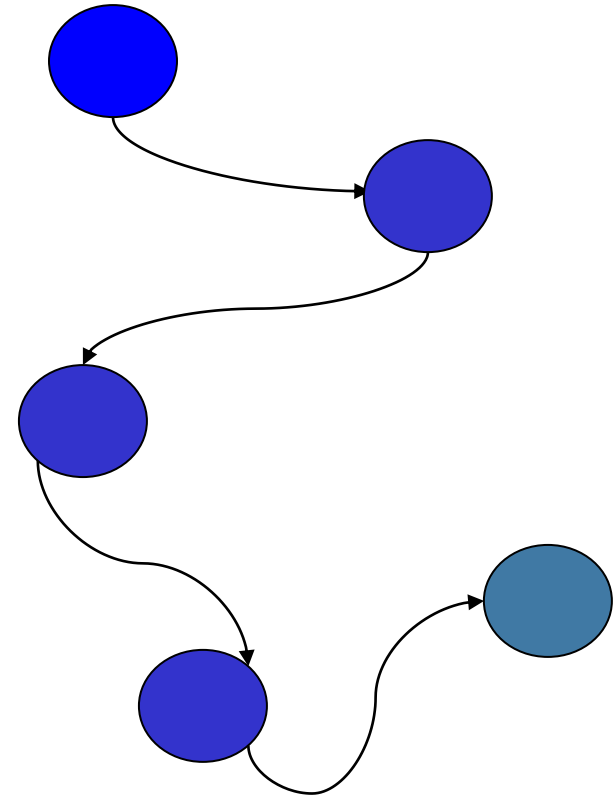
- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

SOAP

- ▶ *Simple Object Accces Protocol*
 - ▶ <http://www.w3.org>
- ▶ SOAP especifica:
 - ▶ 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

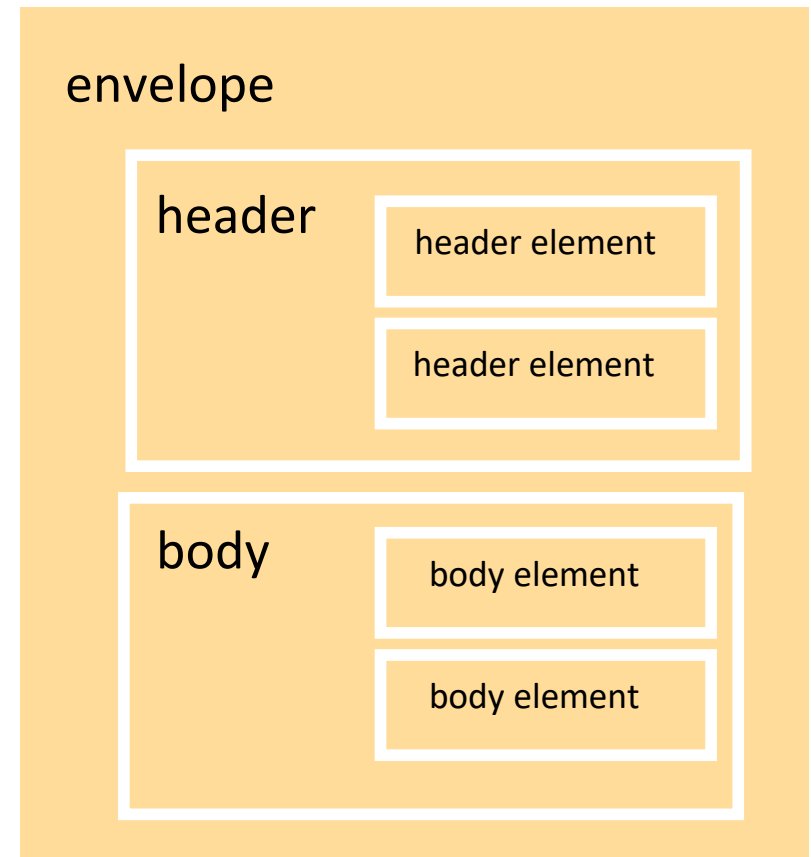
Nodo SOAP

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



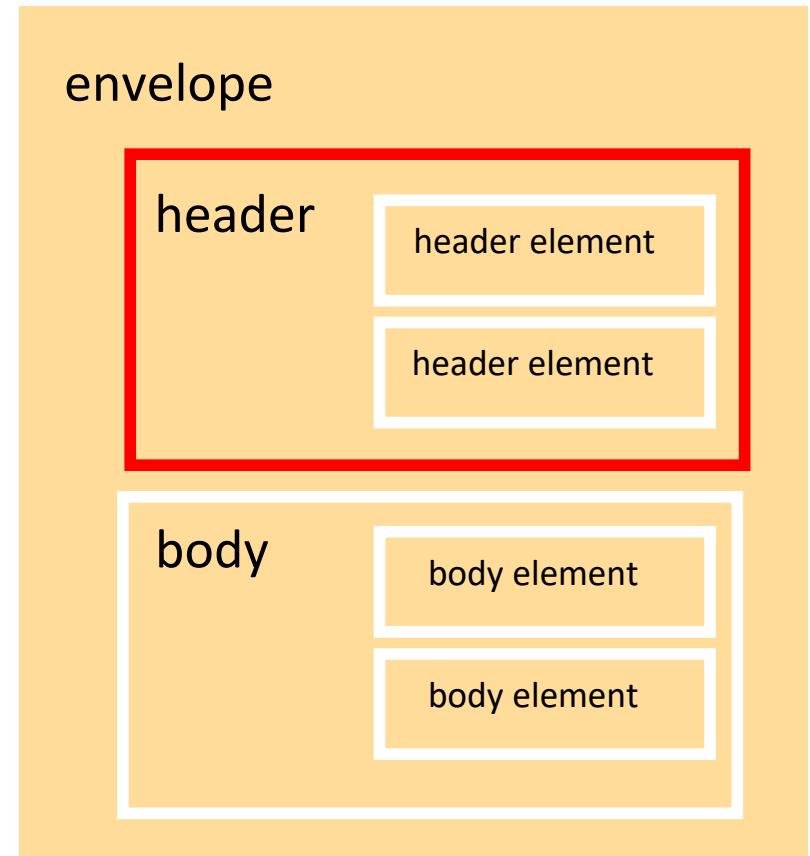
Mensaje SOAP

- ▶ 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>



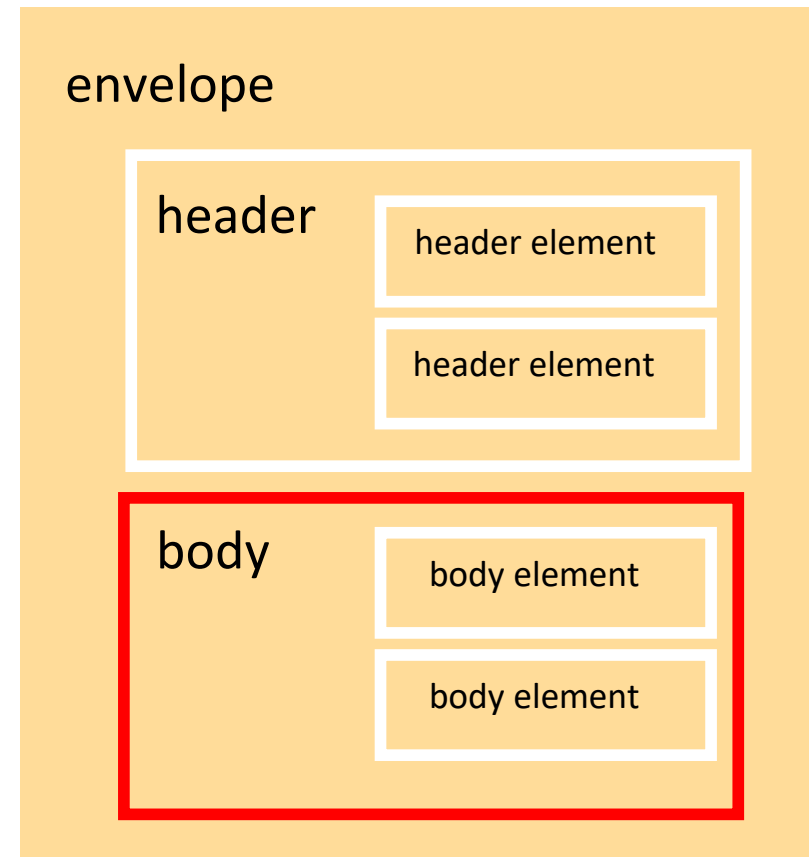
Mensaje SOAP: encabezado

- ▶ 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.

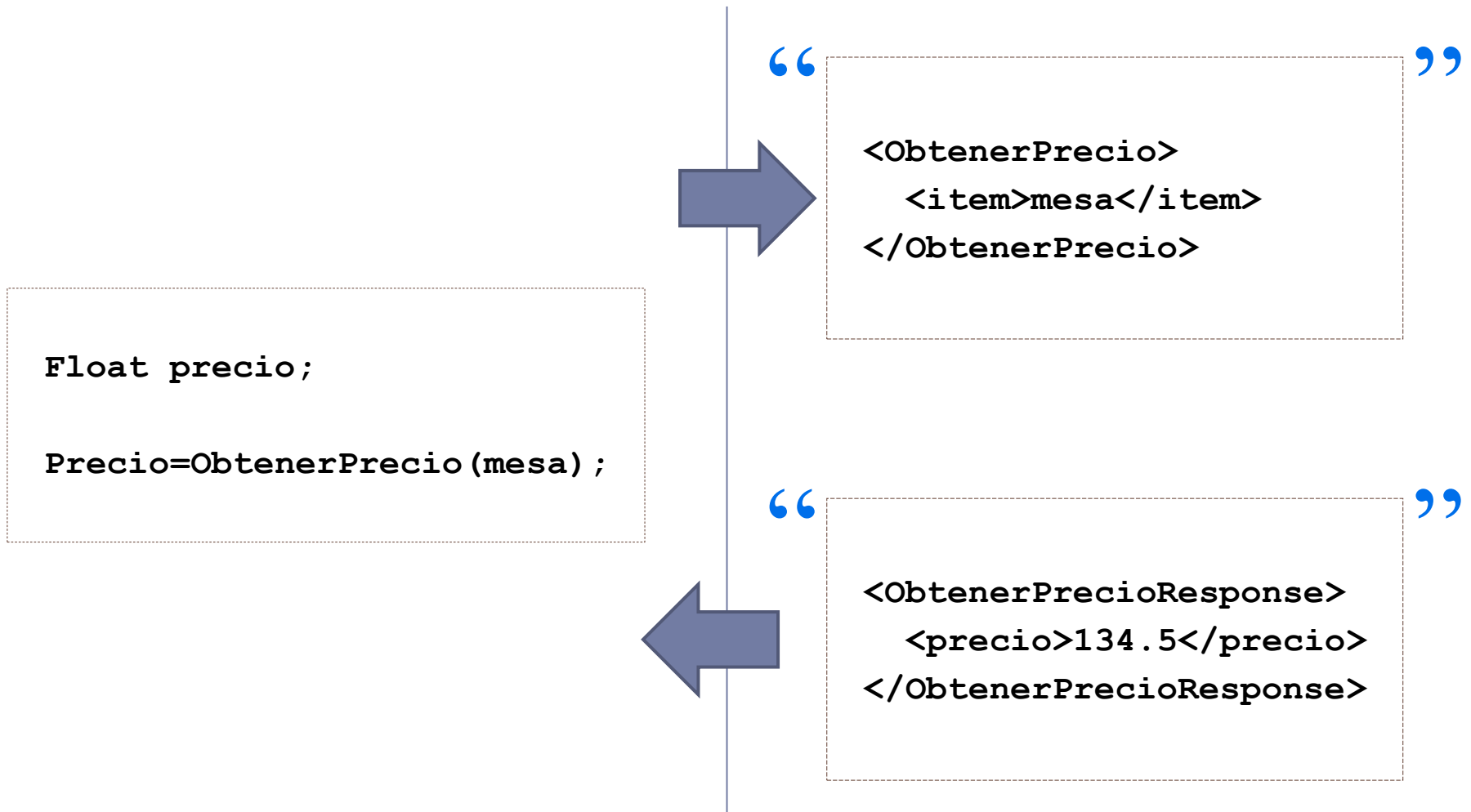


Mensaje SOAP: cuerpo

- ▶ 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



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>
```

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

Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. XML, SOAP, **WSDL**, UDDI

3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

WSDL

- WSDL: *Web Services Description Language*
 - IDL para servicios Web en XML
- Se utiliza para:
 - Describir servicios Web
 - Especifica las operaciones y métodos del servicio
 - Localizar servicios Web
- WSDL es un documento XML
 - Escrito en XML
 - Estándar descrito por la W3C
 - <http://www.w3.org/TR/wsdl>
 - <http://www.w3.org/TR/wsdl20>

Ejemplo de WSDL

string NumberToWords (unsignedLong ubiNum)

- Normalmente generado automáticamente, da lugar a:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.dataaccess.com/webservicesserver/" name="Conversions"
  targetNamespace="http://www.dataaccess.com/webservicesserver/">
  <types>
    <xsd:schema elementFormDefault="qualified" targetNamespace="http://www.dataaccess.com/webservicesserver/">
      <xsd:element name="NumberToWords">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="ubiNum" type="xsd:unsignedLong"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="NumberToWordsResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="NumberToWordsResult" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </types>
  <message name="NumberToWordsSoapRequest">
    <part name="parameters" element="tns:NumberToWords"/>
  </message>
  <message name="NumberToWordsSoapResponse">
    <part name="parameters" element="tns:NumberToWordsResponse"/>
  </message>
  <portType name="ConversionsSoapType">
    <operation name="NumberToWords">
      <documentation>Returns the word corresponding to the positive number passed as parameter. Limited to quadrillions.</documentation>
      <input message="tns:NumberToWordsSoapRequest"/>
      <output message="tns:NumberToWordsSoapResponse"/>
    </operation>
  </portType>
  <binding name="ConversionsSoapBinding" type="tns:ConversionsSoapType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="NumberToWords">
      <soap:operation soapAction="" style="document"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="Conversions">
    <documentation>The Conversion Visual DataPlex Web Service will provide different conversion functions. The function currently available will help you converting numbers into words.</documentation>
    <port name="ConversionsSoap" binding="tns:ConversionsSoapBinding">
      <soap:address location="http://www.dataaccess.com/webservicesserver/conversions.wsdl"/>
    </port>
  </service>
</definitions>
```



Ejemplo de WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.dataaccess.com/webservicesserver/"
  name="Conversions" targetNamespace="http://www.dataaccess.com/webservicesserver/">
  <types>
    <xs:schema elementFormDefault="qualified" targetNamespace="http://www.dataaccess.com/webservicesserver/">
      <xs:element name="NumberToWords">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ubiNum" type="xs:unsignedLong"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="NumberToWordsResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="NumberToWordsResult" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </types>
```

```

<message name="NumberToWordsSoapRequest">
  <part name="parameters" element="tns:NumberToWords"/>
</message>
<message name="NumberToWordsSoapResponse">
  <part name="parameters" element="tns:NumberToWordsResponse"/>
</message>
<portType name="ConversionsSoapType">
  <operation name="NumberToWords">
    <documentation>Returns the word corresponding to the positive number passed as parameter. Limited to
      quadrillions.</documentation>
    <input message="tns:NumberToWordsSoapRequest"/>
    <output message="tns:NumberToWordsSoapResponse"/>
  </operation>
</portType>
<binding name="ConversionsSoapBinding" type="tns:ConversionsSoapType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="NumberToWords">
    <soap:operation soapAction="" style="document"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
<service name="Conversions">
  <documentation>The Conversion Visual DataFlex Web Service will provide different conversion functions. The function currently
    available will help you converting numbers into words.</documentation>
  <port name="ConversionsSoap" binding="tns:ConversionsSoapBinding">
    <soap:address location="http://www.dataaccess.com/webservicesserver/conversions.wso"/>
  </port>
</service>
</definitions>

```

Estructura de un documento WSDL

```
<definitions>
  <types>
    definición de tipos (independientes del lenguajes)
  </types>
  <message>
    definición de mensajes (a intercambiar)
  </message>
  <interfaz>
    definición de puertos (interfaz de funciones, incluyendo parámetros, etc.)
  </portType>
  <binding>
    definición de enlaces (formato de los mensajes y datos a usar)
  </binding>
  <services>
    definición de servicios (nombre de servicio y 1 ó más puertos donde se dá)
  </services>
</definitions>
```

Espacio de nombres

- ▶ Definido por el W3C:
<http://www.w3.org/2001/XMLSchema>
- ▶ **Objetivo:** evitar conflictos
 - ▶ Dos servicios web distintos A y B que tienen un elemento común f.
- ▶ Cada instancia de f se puede referir como A:f o B:f

Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. XML, SOAP, WSDL, **UDDI**

3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

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 actividades
 - ▶ 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

URI, URL y URN

- ▶ Cada servicio Web tiene una **URI** (*Uniform Resource Identifier*):
 - ▶ **URL** (*uniform resource locator*)
 - ▶ Incluyen la localización del recurso (*hostname+pathname*)
Ej.: <https://www.uc3m.es/index.html>
 - ▶ **URN** (*uniform resource name*)
 - ▶ Nombres de recursos que no incluyen localización
Ej.: [doi:10.1016/j.future.2013.01.010](https://doi.org/10.1016/j.future.2013.01.010)
- ▶ Los clientes usan la **URI** para referenciar el servicio
- ▶ Existen servicios de localización de servicios (directorios)
 - ▶ **URN -> URL**
 - ▶ Permiten transparencia de localización

Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

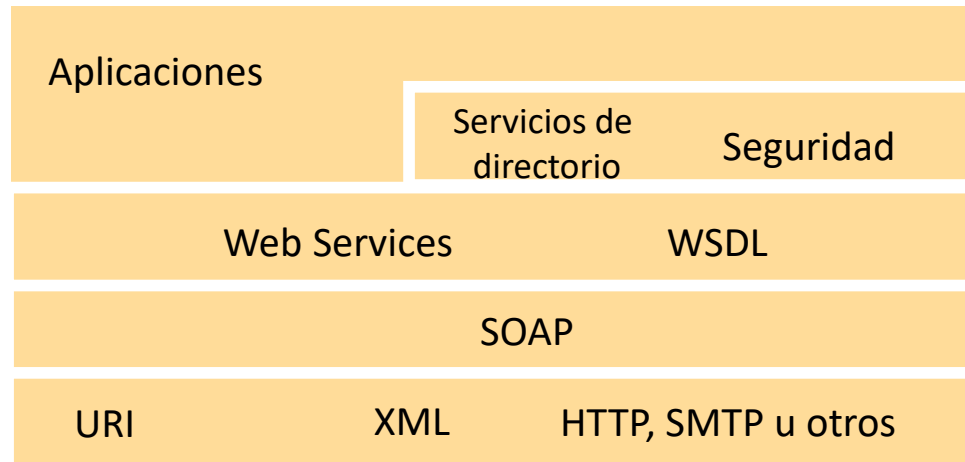
1. XML, SOAP, WSDL, UDDI

3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

Componentes e infraestructura

repaso



- ▶ 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

Más información

- Sobre protocolos:
 - SOAP, XML, etc.: <http://www.w3.org/>
 - UDDI: <http://www.uddi.org/>
- Cursos sobre SOAP, WSDL y otras tecnologías web:
 - <http://www.w3schools.com/>
- Repositorios de servicios Web:
 - <http://www.xmethods.com/>
 - Google: <http://www.google.com/apis>
 - Para aplicaciones que realizan búsquedas en Internet

Entornos de desarrollo

- ▶ Número creciente de entornos de desarrollo
- ▶ Algunas implementaciones de interés:
 - ▶ gSOAP
 - ▶ .Net de Microsoft
 - ▶ Apache Axis2
 - ▶ Java Web Services Developer Pack
 - ▶ IBM WebSphere SDK for Web services (WSDK)
 - ▶ WASP de Systinet
 - ▶ JOnAS
 - ▶ JAX-WS

Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. XML, SOAP, WSDL, UDDI

3. Ejemplo de aplicación

- ▶ **[Python + HTTP] Cliente Web**
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

Cliente HTTP en Python

```
import requests
from bs4 import BeautifulSoup

url = input('Introducir URL de la página: ')
page = requests.get(url)

soup = BeautifulSoup(page.content, 'html.parser')
links = soup.find_all('a')
for a_elto in links:
    print(a_elto)
```

Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. XML, SOAP, WSDL, UDDI

3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ **[Python + SOAP] C de eco (servicio público)**
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

Servicios Web en Python

- ▶ Múltiples entornos:
 - ▶ <https://wiki.python.org/moin/WebServices>
- ▶ Ejemplos:
 - ▶ **Zeep**: para crear **clientes**
 - ▶ Modelo basado en SOAP para Python.
 - ▶ Hace uso de los diccionarios de Python para el manejo de XML.
 - ▶ **Spyne**: para crear **servicios**
 - ▶ Modelo basado en SOAP para Python para creación de servicios.
 - ▶ Despliegue del servidor.
 - ▶ Generador de WDSL.

Zeep

- ▶ Instalar con:

```
pip install zeep          # instalar en el sistema  
pip install zeep --user  # instalar solo usuario actual
```

- ▶ Conocer información del servicio web (esquema, tipos, operaciones) con:

```
python -mzeep <URL al archivo WSDL>
```

Cliente WS en Python de eco

► ws-eco.py

```
import zeep

wsdl = 'http://www.soapclient.com/xml/soapresponder.wsdl'
client = zeep.Client(wsdl=wsdl)
print(client.service.Method1('prueba', 'WS'))
```

► Salida

Your input parameters are **prueba** and **WS**

Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. XML, SOAP, WSDL, UDDI

3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ **[Python + SOAP] C+S de calculadora (privado)**
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ [C + SOAP] C+S de calculadora (privado)

Spyne

► Instalar con:

```
pip install spyne          # instalar en el sistema  
pip install spyne --user  # instalar solo usuario actual
```

Servidor de calculadora...

► ws-calc-service.py

```
import time, logging
from wsgiref.simple_server import make_server
from spyne import Application, ServiceBase, Integer, Unicode, rpc
from spyne.protocol.soap import Soap11
from spyne.server.wsgi import WsgiApplication

class Calculadora(ServiceBase):
    @rpc(Integer, Integer, _returns=Integer)
    def add(ctx, a, b):
        return a+b
    @rpc(Integer, Integer, _returns=Integer)
    def sub(ctx, a, b):
        return a-b

application = Application(services=[Calculadora], tns='http://tests.python-zeep.org/',
                          in_protocol=Soap11(validator='lxml'), out_protocol=Soap11())
application = WsgiApplication(application)

if __name__ == '__main__':
    logging.basicConfig(level=logging.DEBUG)
    logging.getLogger('spyne.protocol.xml').setLevel(logging.DEBUG)
    logging.info("listening to http://127.0.0.1:8000; wsd1 is at: http://localhost:8000/?wsdl")
    server = make_server('127.0.0.1', 8000, application)
    server.serve_forever()
```

Servidor de calculadora...

► Arrancar servicio

```
python3 ws-calc-service.py &  
python3 -mzeep http://localhost:8000/?wsdl
```

► Salida

Operations:

```
add(a: xsd:integer, b: xsd:integer) -> addResult: xsd:integer  
sub(a: xsd:integer, b: xsd:integer) -> subResult: xsd:integer
```

Cliente de calculadora...

► ws-calc-client.py

```
import zeep

wsdl = "http://localhost:8000/?wsdl"
client = zeep.Client(wsdl=wsdl)

r = client.service.add(5, 2)
print(" 5+2 = " + str(r))
r = client.service.sub(5, 3)
print(" 5-3 = " + str(r))
```

Cliente de calculadora...

▶ Arrancar cliente

```
python3 ws-calc-client.py
```

▶ Salida

```
7
```

```
2
```


Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. XML, SOAP, WSDL, UDDI

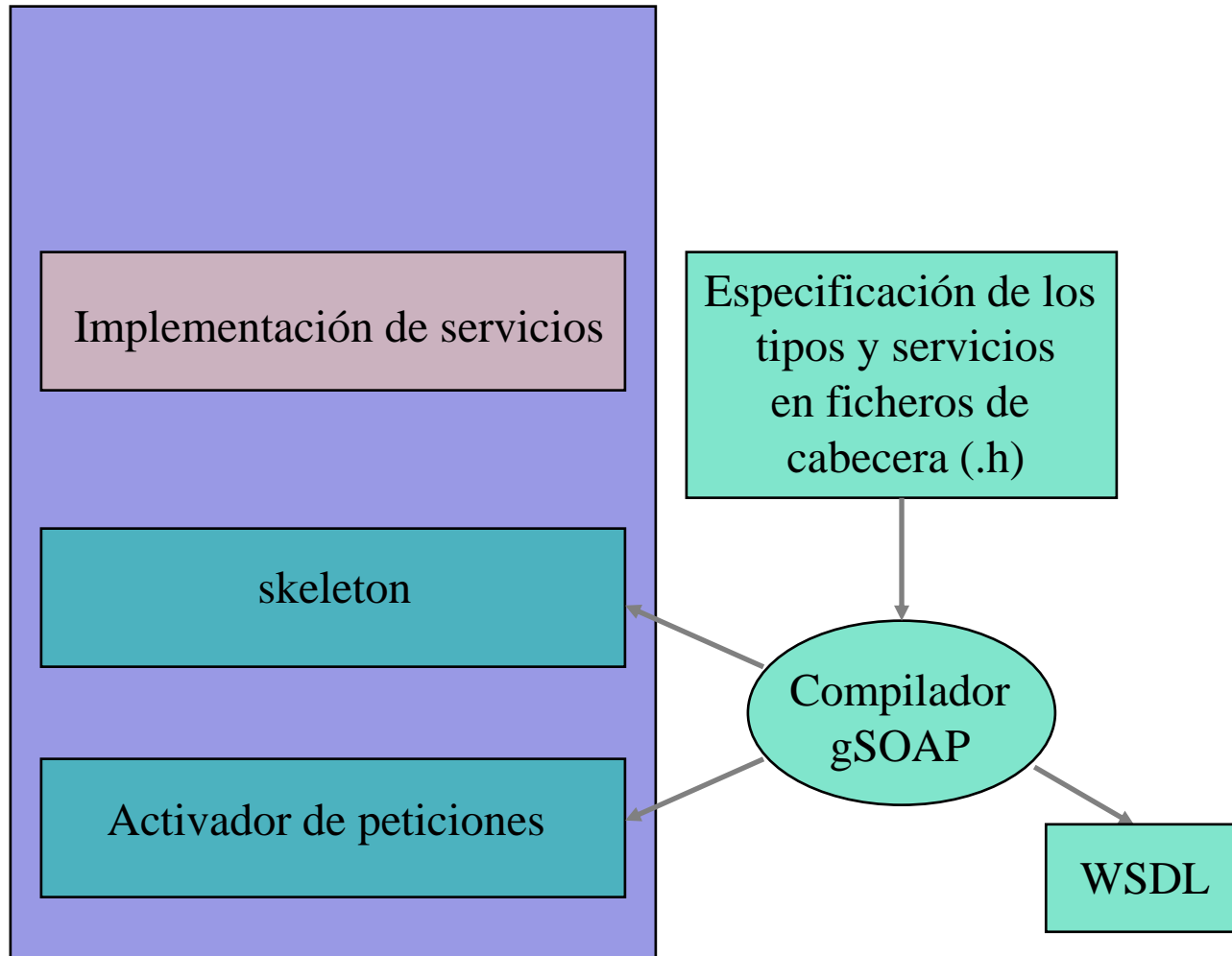
3. Ejemplo de aplicación

- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ **[C + SOAP] C de calculadora (servicio público)**
- ▶ [C + SOAP] C+S de calculadora (privado)

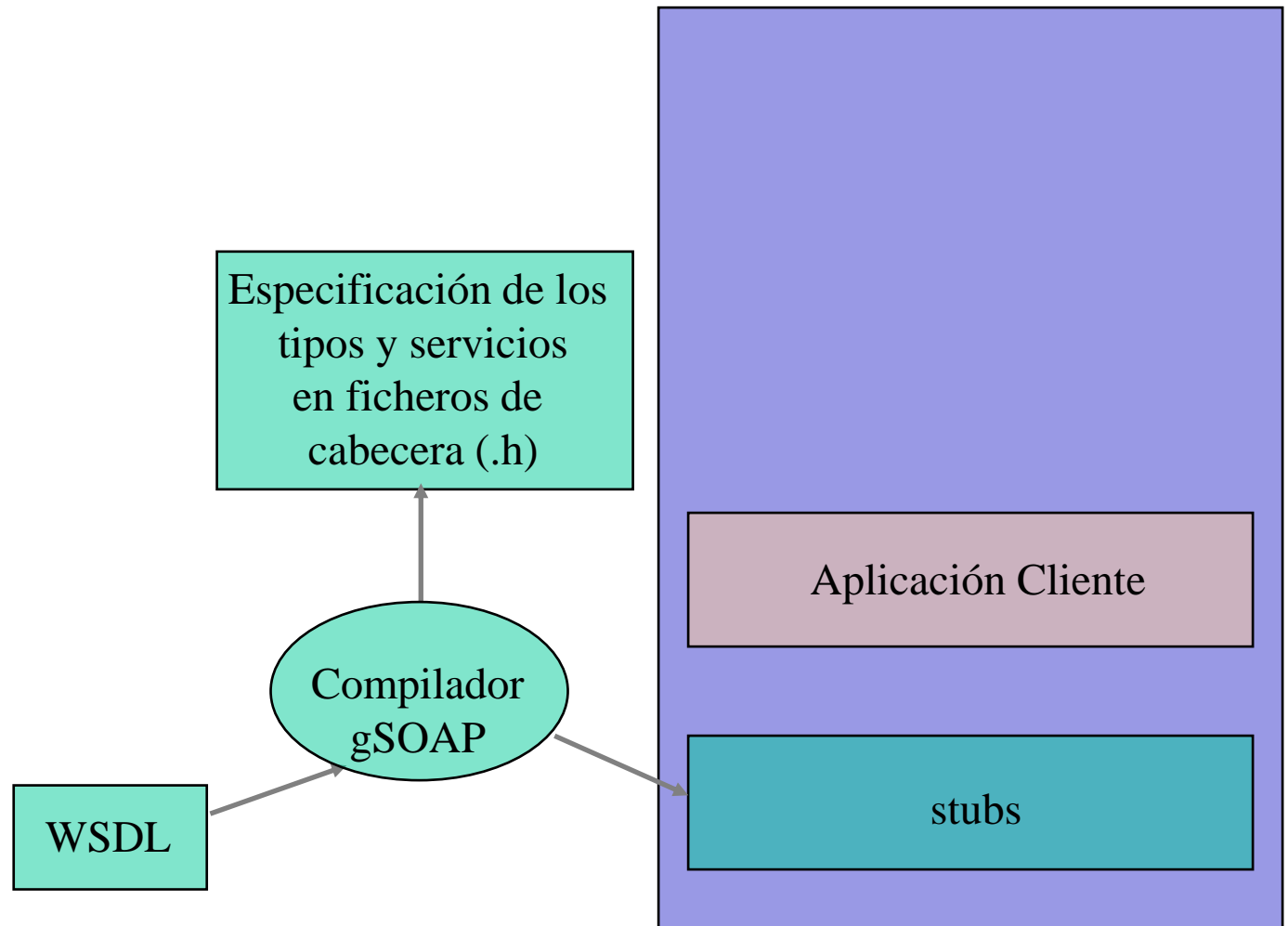
Plataforma de desarrollo

- ▶ 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 del servidor



Desarrollo del cliente



Ejemplo: calculadora

- ▶ WSDL:

- ▶ <http://www.dneonline.com/calculator.asmx?WSDL>

- ▶ Accesible desde:
<http://www.xmethods.com/>

- ▶ Descrito en:
<http://www.dneonline.com/calculator.asmx>

- ▶ Tres métodos:

```
int Add ( int a, int b )  
// Adds two integers.This is a test WebService. ©DNE Online
```

Generación de la interfaz a partir del WSDL

```
acaldero@guernika# wsd12h -c \  
                        -o calc.h \  
                        http://www.dneonline.com/calculator.asmx?WSDL
```

Preprocesado de la interfaz a C

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

Client.c

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

int main ( int argc, char *argv[] )
{
    struct soap *soap;
    int ret;
    struct __tempuri__Add      args ;
    struct __tempuri__AddResponse res ;

    soap = soap_new();
    if (NULL == soap) {
        return -1;
    }

    args.intA = 1;
    args.intB = 2;
    ret = soap_call__tempuri__Add(soap, NULL, NULL, &args, &res) ;
    if (SOAP_OK != ret) {
        soap_print_fault(soap, stderr);
        return -1;
    }

    printf("Sum = %d\n", res.AddResult);

    soap_destroy(soap);
    soap_end(soap);
    soap_free(soap);

    return 0;
}
```


Compilación del ejemplo

guernika.lab.inf.uc3m.es

```
# gcc -g -c soapC.c          -o soapC.o
# gcc -g -c soapClientLib.c  -o soapClientLib.o
# gcc -g -c app-d.c           -o app-d.o

# gcc -o app-d -g app-d.o soapClientLib.o soapC.o -lgsoap
```

Ejecución del ejemplo

guernika.lab.inf.uc3m.es

```
acaldero@guernika # ./client
```

```
Sum = 3
```

Contenidos

1. Introducción:

1. Paradigma de servicios de red
2. Servidor Web también para llamadas remotas

2. Elementos en un Servicio Web:

1. XML, SOAP, WSDL, UDDI

3. Ejemplo de aplicación

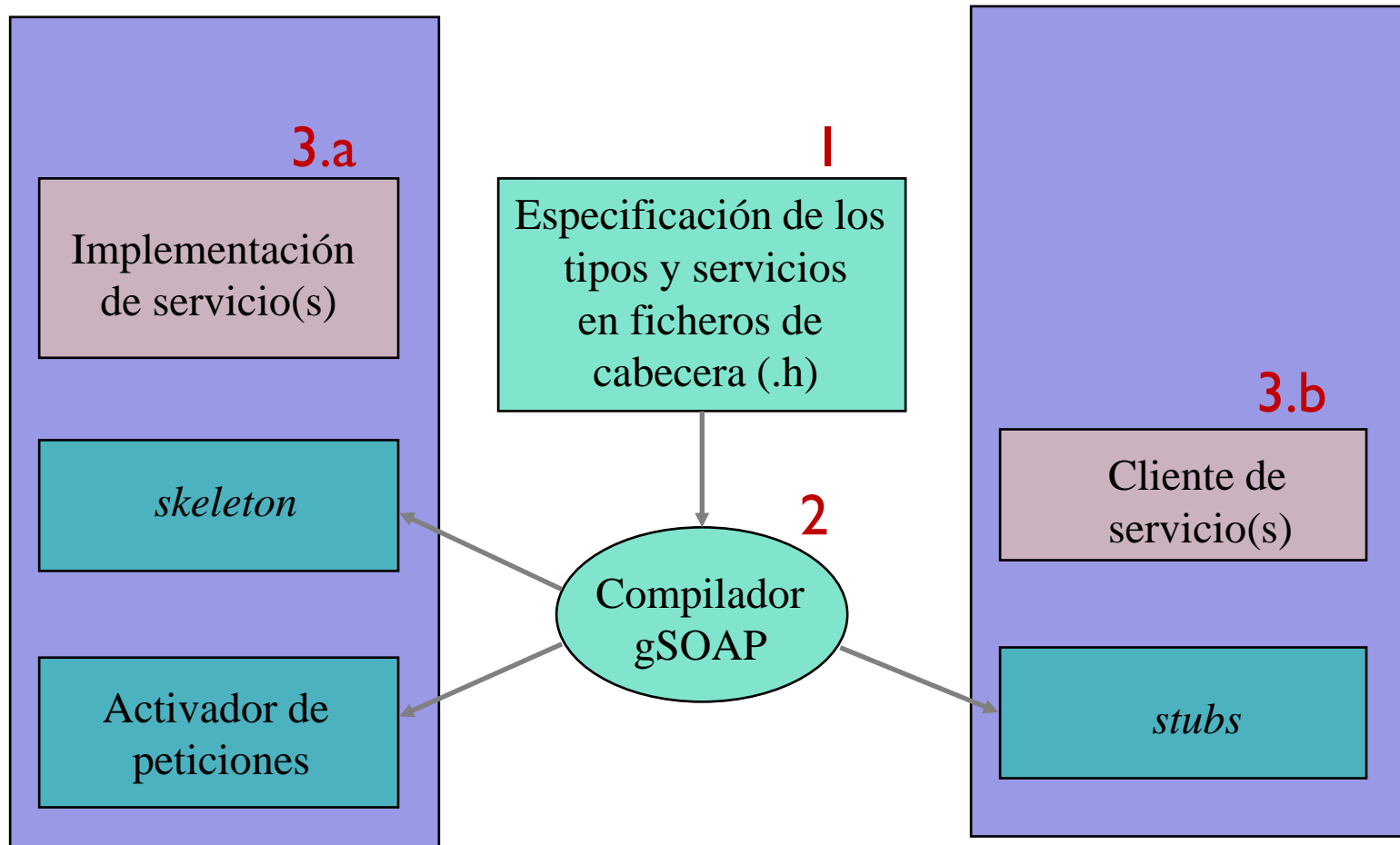
- ▶ [Python + HTTP] Cliente Web
- ▶ [Python + SOAP] C de eco (servicio público)
- ▶ [Python + SOAP] C+S de calculadora (privado)
- ▶ [C + SOAP] C de calculadora (servicio público)
- ▶ **[C + SOAP] C+S de calculadora (privado)**

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.nsmmap"

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


Ejecución del ejemplo

guernika.lab.inf.uc3m.es

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

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

Grupo ARCOS



uc3m | Universidad **Carlos III** de Madrid

Tema 8: Servicios Web

Sistemas Distribuidos



Grado en Ingeniería Informática