Tema 8
Servicios web

Sistemas Distribuidos
Grado en Ingeniería Informática
Universidad Carlos III de Madrid

Contenido

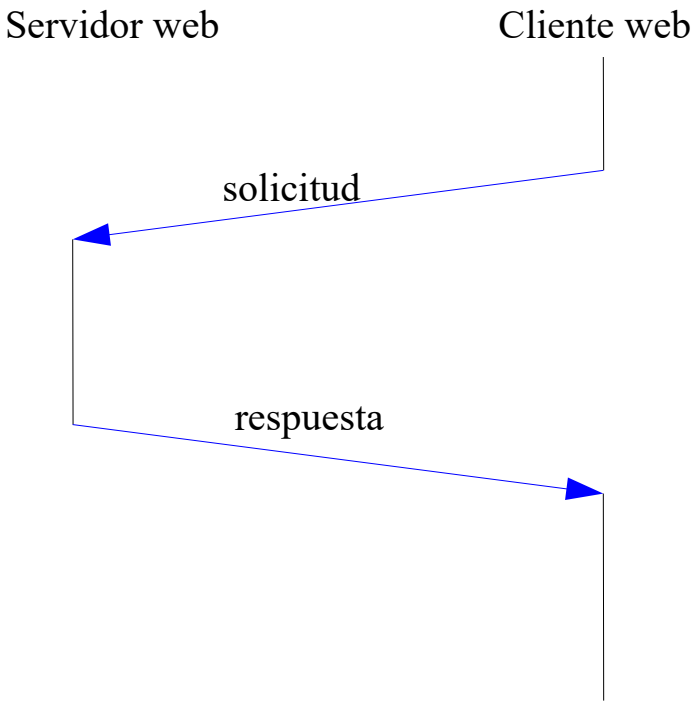
- Protocolo HTTP
- Clientes y servidores web
- Servicios web
- Principios básicos de diseño
- SOAP
- REST
- Entornos de desarrollo

Introducción

- Elementos que conforman la World Wide Web (WWW):
  - Documentos hipertexto
  - Protocolo HTTP
  - HTML
  - Servidor Web
  - Navegadores

Protocolo HTTP

- HyperText Transfer Protocol se usa en WWW para transferir hipertexto (páginas HTML con hiperenlaces)
  - Usa el **puerto TCP 80** para aceptar conexiones entrantes
- Se basa en el paradigma cliente-servidor



Elementos solicitud:

- <mandato> <dirección documento > <versión HTTP >
- cabecera opcional
- datos opcionales

Elementos respuesta:

- línea de estado con formato <protocolo><códigos estado><descripción>
- información de cabecera
- documento.

Ejemplo

```
[redacted]~$ telnet www.uc3m.es 80
Trying 176.58.10.138...
Connected to www.uc3m.es.
Escape character is '^]'.
GET /
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose
.dtd">
<html>
<head>
<title>UC3M</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<noscript>
        <meta http-equiv="refresh" content="0; URL=https://www.uc3m.es/Inicio" />
</noscript>

<script type="text/javascript">

        var idiomaNavegador = navigator.language ? navigator.language : navigator.userLanguage

        function redirectPage(dest){
                if (window.location.replace)
                        window.location.replace(dest)
                else
                        window.location = dest
        }

        if (idiomaNavegador != null && idiomaNavegador.toLowerCase().substr(0,2)=="es") {
                redirectPage("https://www.uc3m.es/Inicio")
        } else {
                redirectPage("https://www.uc3m.es/Home")
        }
</script>

</head>
<body>
</body>
</html>
```

Protocolo HTTP: petición

- Se establece una conexión al host al puerto 80
- Línea de petición:

```
<Método><espacio><URI solicitado><espacio><protocolo>\r\n
```

  - Donde **método**:
    - GET: solicita una página WEB
    - HEAD: solicita la cabecera de una página Web
    - POST: envía datos a una aplicación Web
    - PUT: solicita almacenar una página web
  - donde **URI** (Uniform Resource Identifier)
    - URL Uniform Resource Locator (http://www.uc3m.es)
    - URN Uniform Resource Name (doi:10.1016/j.future.2013.01.010)
  - donde **protocolo**
    - HTTP/0.9
    - HTTP/1.0
    - HTTP/1.1
    - HTTP/1.2

Protocolo HTTP: respuesta

- Contenido de la respuesta:

```
<protocolo> <código>
<cabeceras>
<recurso>
```

donde protocolo es aquel que entiede el servidor

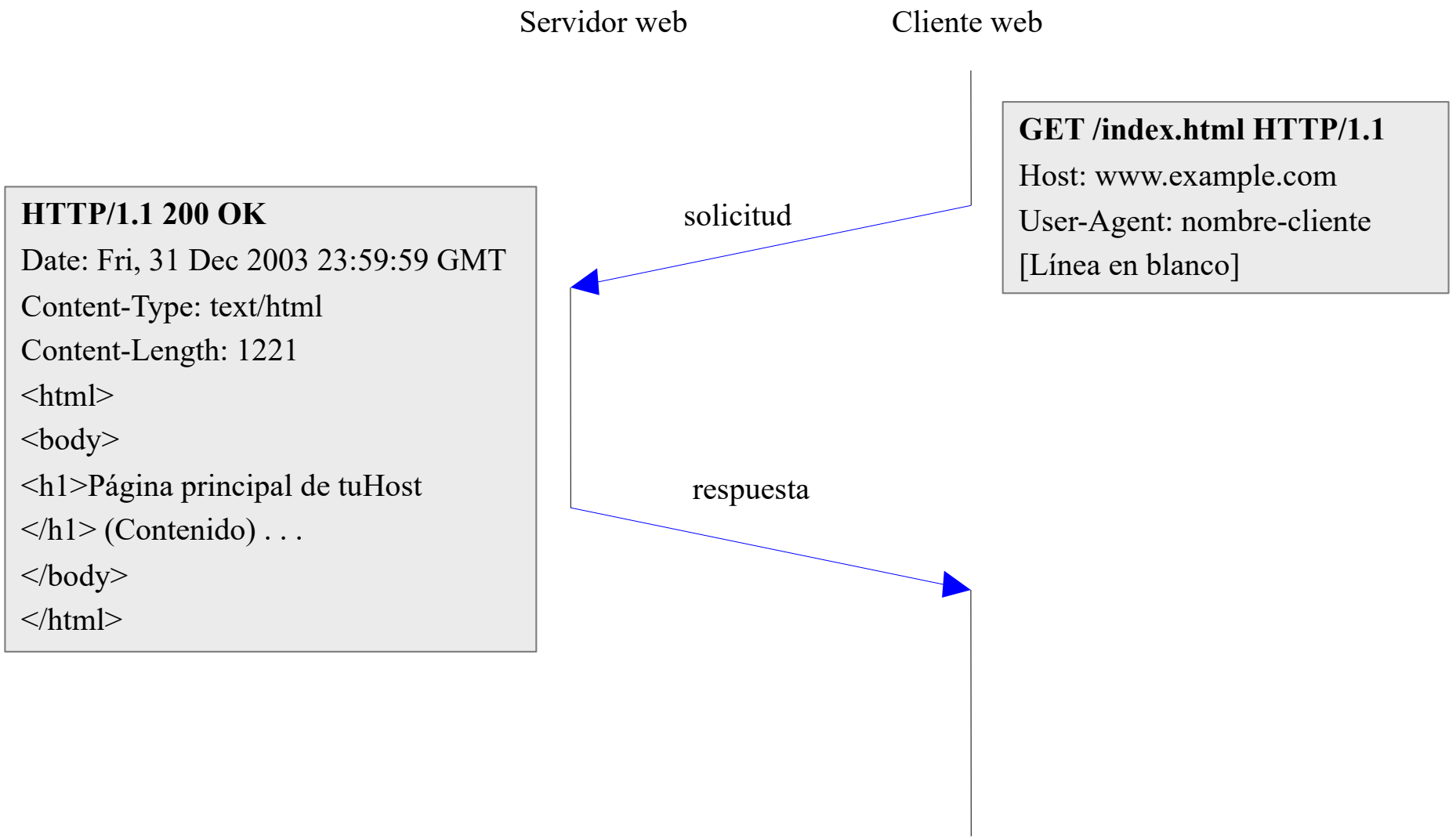
- HTTP/0.9
- HTTP/1.0
- HTTP/1.1
- HTTP/1.2

donde código es un código de error:

- 200 → OK
- 400 → Error en el cliente
- 500 → Error en el servidor

…..

Ejemplo


Servidor web
Cliente web
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: nombre-cliente
[Línea en blanco]
solicitud
HTTP/1.1 200 OK
Date: Fri, 31 Dec 2003 23:59:59 GMT
Content-Type: text/html
Content-Length: 1221
<html>
<body>
<h1>Página principal de tuHost
</h1> (Contenido) . . .
</body>
</html>
respuesta

Protocolo HTTP: servidor

- Contenido de la respuesta:

```
HTTP/1.1 200 OK
```

Línea

```
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
```

Cabeceras

```
<html>
<head>
<title>My web page </title>
</head>
<body>
Hello world!
</BODY></HTML>
```

Recurso

Tipos de páginas WEB

- Páginas web estáticas
- Páginas web dinámicas
  - Ejecutadas en el cliente
    - JavaScript
    - Applet
  - Ejecutadas en el servidor
    - CGI (*Common Gateway Interface*)
    - PHP
    - ASP
    - Servlet

Cliente HTTP en Python

```
import requests

url = input('Webpage to grab source from: ')

req = requests.get(url, 'html.parser')

print(req.text)
```

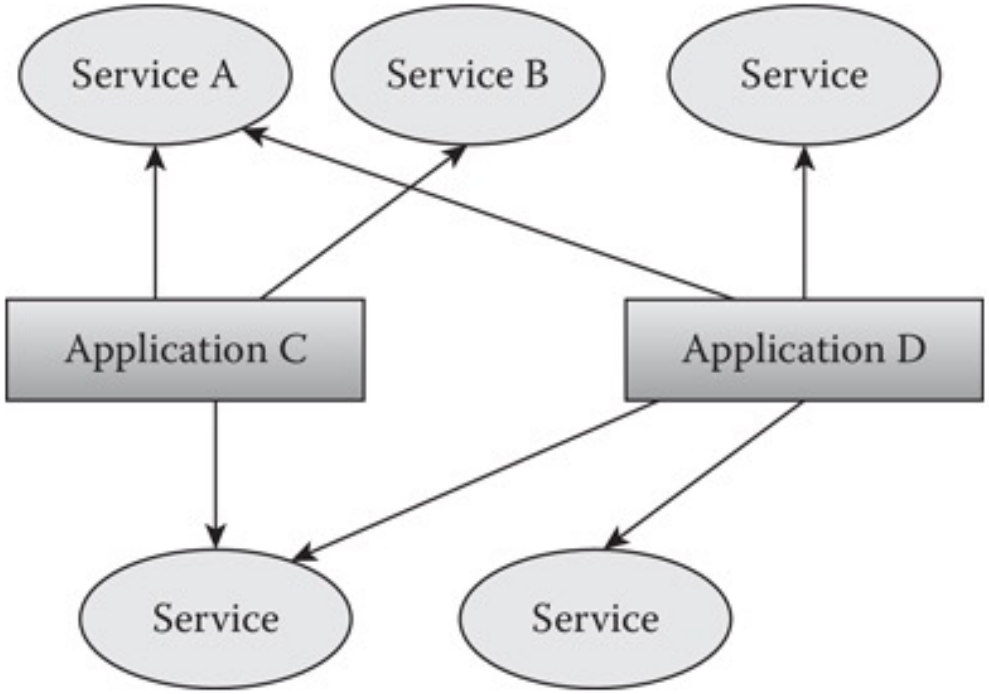
Evolución de la arquitectura de las aplicaciones

- Monolíticas
- Cliente-setvidor
- Arquitecturas 3-tier N-tier
- Objetos distribuidos
- Arquitecturas orientadas a servicios

SOA: Service Oriented Architecture

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

Arquitectura orientada a servicios



©Introduction to Middleware, Web Services, Object Components and Cloud Computing

Tipos de servicios

- Sin estado: peticiones de servicio autocontenidas
  - Servicios más fiables y sencillos
  - Servidores más escalables
- Con estado: almacenan estado estableciendo una sesión entre consumidor y proveedor del servicio
  - Razones de eficiencia, personalización de servicios
  - Incrementa el acoplamiento entre el cliente y el servidor
  - Puede reducir la escalabilidad del servidor
- Peticiones idempotentes: no realizan ningún cambio. Peticiones duplicadas tienen el mismo efecto que una única petición. Incrementan la fiabilidad, repitiendo la petición se hay algún fallo.

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)

Servicios Web

- **Idea:** Adaptar el modelo de programación web (débilmente acoplado) para su uso en aplicaciones no basadas en navegador
- El objetivo es ofrecer una plataforma para construir aplicaciones distribuidas orientadas a servicios utilizando un software que enmascare la heterogenidad
- Un **servicio web** es una colección de protocolos y estándares abiertos que sirven para intercambiar datos entre aplicaciones:
  - Escritas en distintos lenguajes de programación
  - Ejecutan en distintos sistemas operativos y arquitecturas
  - Desarrolladas de manera independiente
- Los servicios web son independientes de la aplicación que los usa
- Estandarización controlada por un grupo del **W3C**:
  - http://www.w3.org/2002/ws/

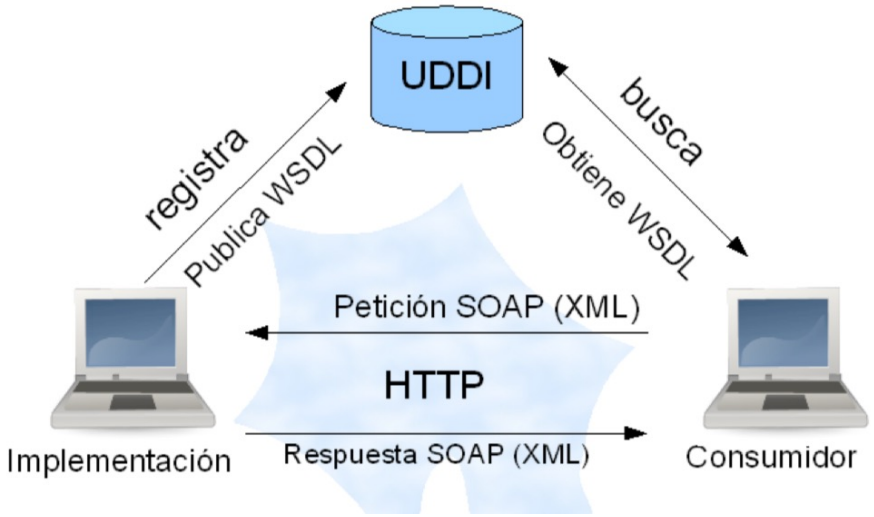
Ventajas e inconvenientes

- ✓ Interoperabilidad entre aplicaciones de SW que pueden ejecutar sobre distintas plataformas
- ✓ Al ejecutar HTTP, pueden atravesar firewalls sin necesidad de cambiar las reglas de filtrado
- ✓ Independencia entre el servicio web y la aplicación que lo usa
- ✓ Fomentan el uso de estándares abiertos

- ✓ Peor rendimiento comparado con otros modelos de computación distribuida: RMI, Corba o DCOM.
- ✓ Pueden esquivar firewalls

Servicios Web SOAP

- Los mensajes se transportan (a excepción de binarios) utilizando el protocolo SOAP
- La descripción del servicio se realiza en WSDL
- Uso de UDDI, que son las siglas del catálogo de negocios de Internet denominado Universal Description, Discovery and Integration.



Interfaz y operaciones

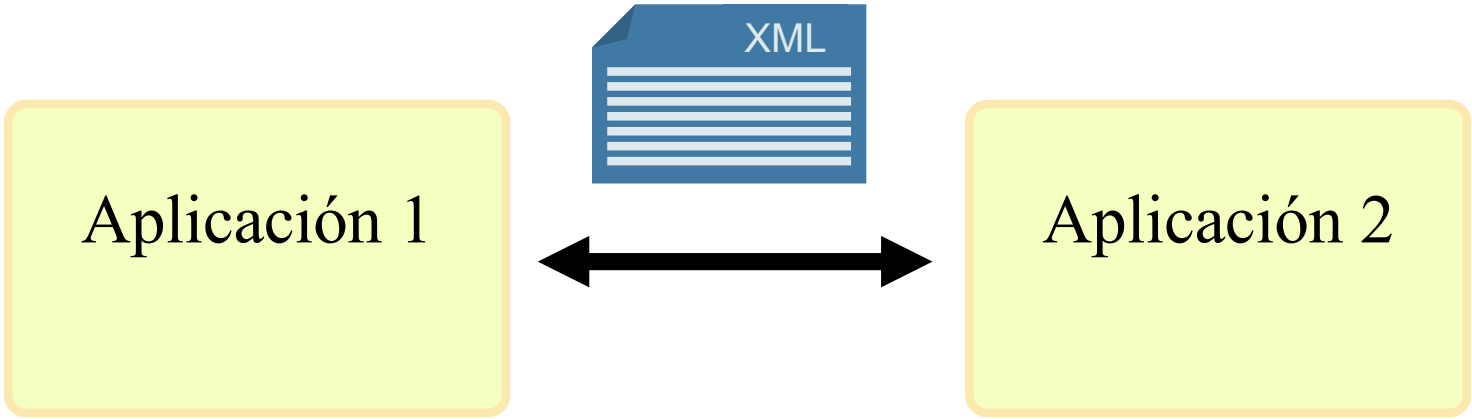
- Una **interfaz** de servicio web consta de un conjunto de operaciones que pueden ser accedidas por un cliente en Internet
  - Los servicios web **no** son específicos de HTTP
- El conjunto de operaciones en un servicio web pueden ser ofrecidas por programas, objetos, bases de datos, etc.
- Un servicio web puede ser manejado por:
  - Un servidor web tradicional
  - Un servidor independiente *(stand-alone)*

Interoperabilidad en entornos heterogéneos

- Servicios basados en protocolos abiertos y estándar
  - Protocolo del nivel de aplicación para la transferencia de mensajes (ej. HTTP)
  - **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 (lenguaje de interfaz)
- Ventajas:
  - Paso de cortafuegos
  - Difícil en otros entornos como Java RMI o CORBA

Representación de mensajes

- Mensajes **SOAP** y datos representados en **XML**



XML

- **eXtensible Markup Language (XML)**
  - Definido por W3C (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>*
    - *Ejemplo:* http://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL
  - Espacios de nombres
    - *xmlns="http://www.w3.org/1999/xhtml"*
  - Esquemas
    - Elementos y atributos que pueden aparecer en un documento

SOAP

- **Simple Object Access Protocol** (SOAP) es un protocolo estandarizado por W3C para el intercambio de mensajes basados en XML

- Usa un protocolo de la capa de aplicación como protocolo de transporte
  - Típicamente el protocolo **HTTP pero** no el único
- El protocolo SOAP soporta distintos patrones de mensajes:
  - El más usado es el **basado en RPC**
- Además usa **WSDL** (Web Service Description Language) para describir web services:
  - Se usa en combinación con XML para proporcionar servicios web sobre Internet

SOAP

- SOAP especifica:
  - Cómo representar los **mensajes de texto** en XML
  - Cómo combinar **mensajes SOAP** para un modelo petición-respuesta
  - Cómo procesar los **elementos de los mensajes**
  - Cómo utilizar el **protocolo de aplicación** (**HTTP, SMTP**, …) para enviar mensajes SOAP

Ejemplo: SOAP request

- **POST / engelen/calcserver.cgi HTTP/1.1**
  Host: websrv.cs.fsu.edu
  User-Agent: gSOAP/2.7
  Content-Type: text/xml; charset=utf-8
  Content-Length: 464
  Connection: close
  SOAPAction: ""

línea de peticion

Cabeceras HTTP

Cabecera HTTP

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:c="urn:calc">
  <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <c:add>
      <a>1</a>
      <b>2</b>
    </c:add>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Envelope

Cabeceras SOAP

Cuerpo SOAP

Cuerpo HTTP

Ejemplo: SOAP response

- **HTTP/1.1 200 OK**
  Date: Wed, 05 May 2010 16:02:21 GMT
  Server: Apache/2.0.52 (Scientific Linux)
  Content-Length: 463
  Connection: close
  Content-Type: text/xml; charset=utf-8

línea de respuesta

Cabeceras HTTP

Cabecera HTTP

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:calc">
  <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns:addResponse>
      <result>3</result>
    </ns:addResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Envelope

Cabeceras SOAP

Cuerpo SOAP

Cuerpo HTTP

Identificación de servicios

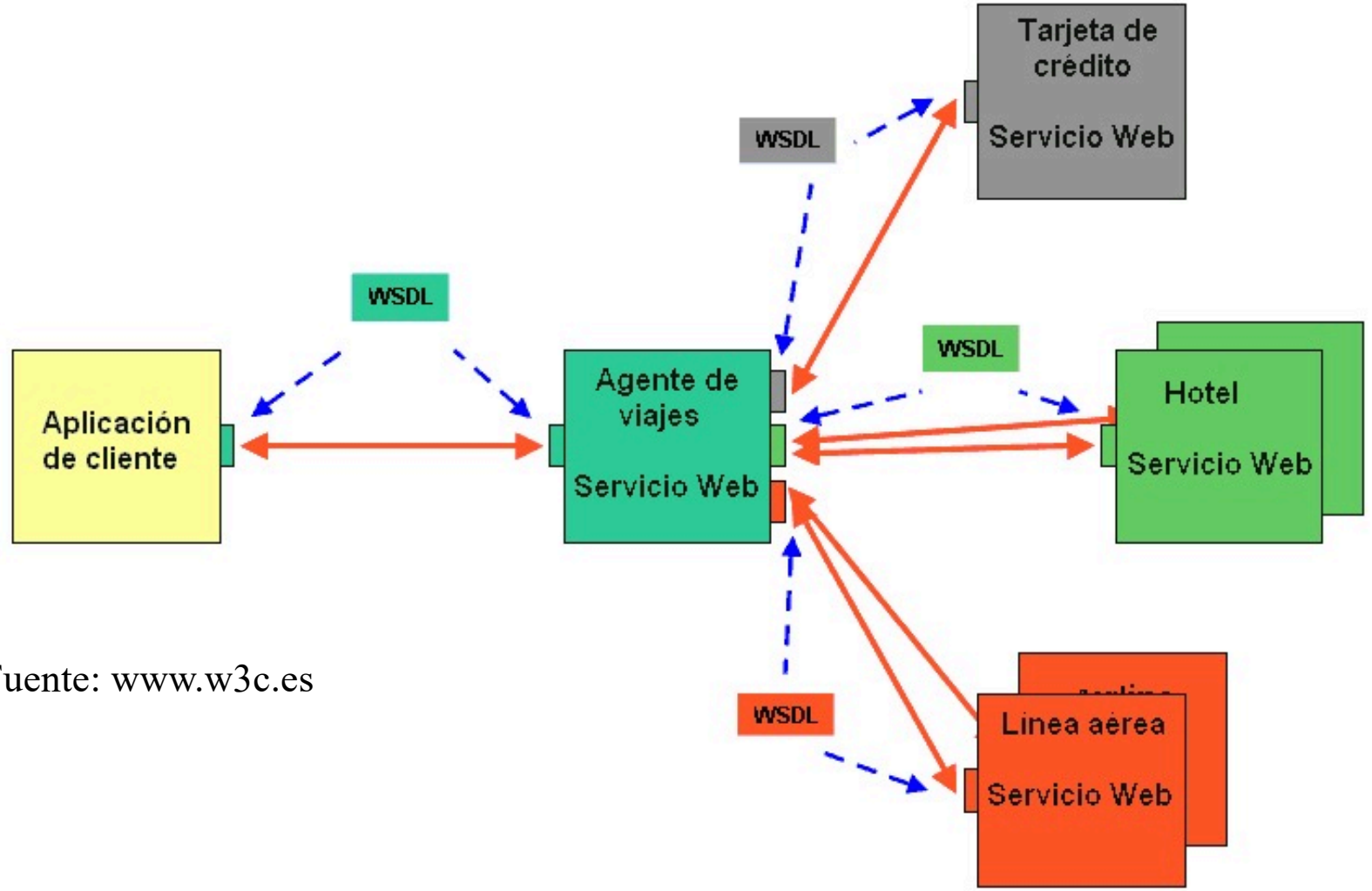
- Cada servicio Web contiene una **URI** (Uniform Resource Identifier) que identifica unívocamente un recurso
  - **URL** (Uniform Resource Locator)
    - Especificado en **RFC 1630**, **1738**, **1808**
    - Incluyen la localización del recurso
    - El formato general de una URL es:
      esquema://máquina[:puerto]/directorio/archivo
    - Ejemplo: http://www.arcos.inf.uc3m.es/~infosd
  - **URN** (Uniform Resource Name)
    - Especificado en **RFC 2141**
    - Nombres de recursos que no incluyen su localización
    - <URN> ::= "urn:" <NID> ":" <NSS>
    - Ejemplo: urn:issn:0167-6423
- Los clientes usan la **URI** para referenciar el servicio

Activación de servicios

- Tipos de activación
  - El servicio web se activa bajo demanda
  - El servicio web ejecuta continuamente

- El servicio web se solicita al computador identificado en la URL
  - Ej: http://**www.arcos.inf.uc3m.es/**~infosd
  - El servicio Web puede residir en ese computador o en otro computador
    - Combinación de servicios web
    - Mejora las prestaciones

Combinación de servicios Web



Fuente: www.w3c.es

WSDL

- **Web Services Description Language**
  - Lenguaje de descripción de interfaz (IDL) para servicios Web en XML
- Describe el protocolo de aplicación
  - Intercambio de los mensajes
  - Formato de los mensajes
- WSDL es un documento escrito en XML
- Se utiliza para:
  - **Describir** servicios Web
  - **Localizar** servicios Web
- WSDL todavía no es un estándar del W3C (*draft*)

Descripción de servicios web

- **Describir** el servicio Web
- **Especificar** la localización del servicio
- **Especificar** las operaciones y métodos del servicio web

- **Normalmente**, generado automáticamente a partir del código fuente del servicio

Estructura de un documento WSDL

<definitions>
    <types>
        definición de tipos (independientes del lenguajes)
    </types>
    <message>
        definición de mensajes (a intercambiar)
    </message>
    <portType>
        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 da)
    </services>
</definitions>

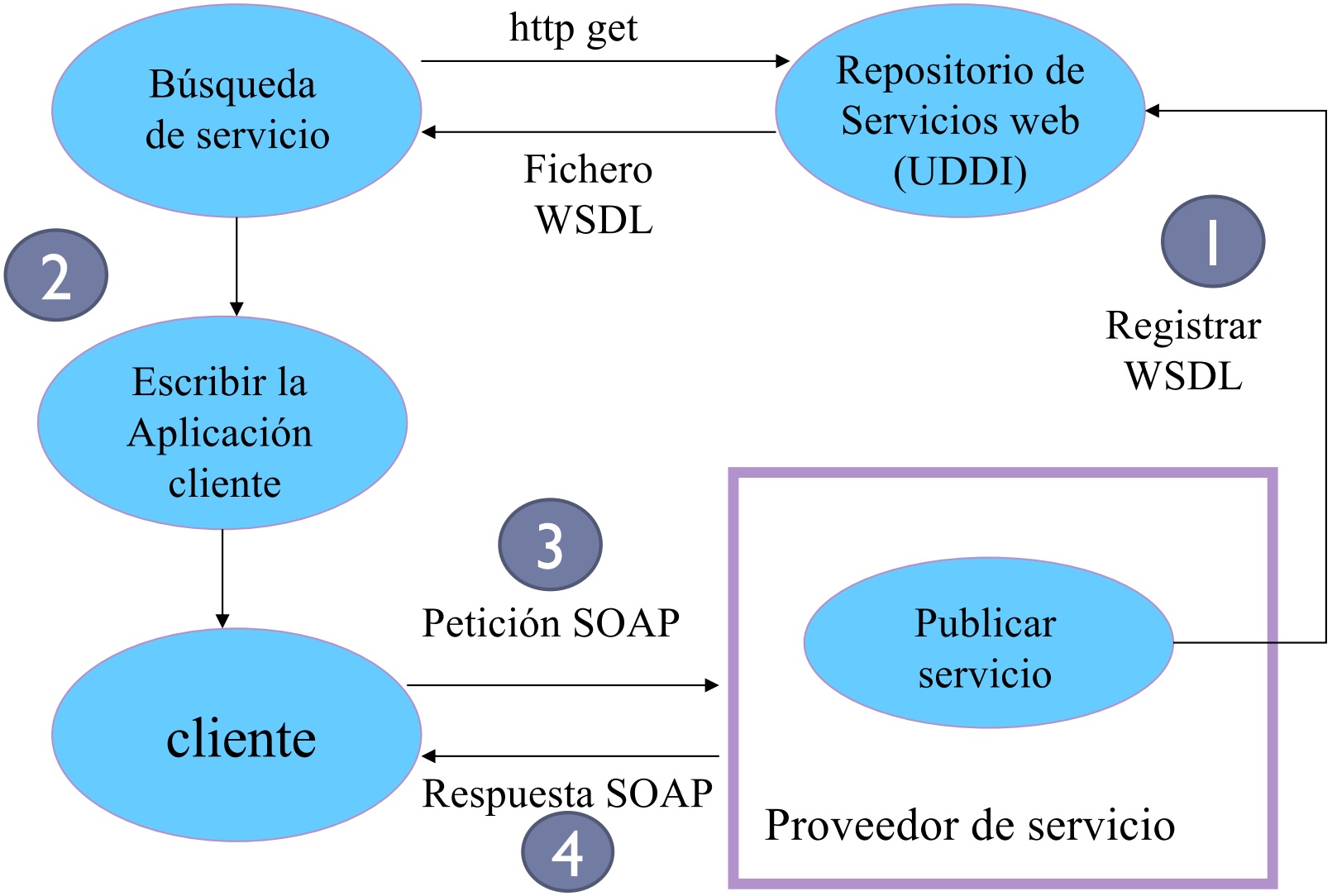
Elementos

- **Types:** tipos independientes del lenguaje
- **Messages:** tipos de mensajes a intercambiar
- **PortTypes (interfaz):** define la interfaz de funciones (nombre de la operación, parámetros de entrada, parámetros de salida)
- **Bindings:** especifica el formato de los mensajes y de los datos a ser utilizados.
- **Services:** especifica el nombre del servicio y uno o más sitios *(puertos)* donde encontrar el servicio..

UDDI

- **Universal Description, Discovery, and Integration**
  - No estándar: Propuesta inicial de Microsoft, IBM y Ariba
- Catálogo de negocios de Internet
  - 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, etc.

Escenario de uso



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*"

Entornos de desarrollo

- Algunas implementaciones de interés:
  - JAX-WS
  - gSOAP
  - .Net de Microsoft
  - Apache Axis2
  - IBM WebSphere SDK for Web services
  - WASP de Systinet
  - JOnAS
  - AXIS

Servicios web en Python

- Múltiples entornos:
- https://wiki.python.org/moin/WebServices
- Ejemplos:
  - Zeep: para crear clientes
  - Spyne: para crear servicios

Zeep

- Modelo basado en SOAP para Python.
- Hace uso de los diccionarios de Python para el manejo de XML.

Instalación

- Instalación en el sistema:
  - pip install zeep

- Instalación para un usuario:
  - pip install zeep --user

Ejemplo básico: servicio web de Eco

```
python -mzeep
http://www.soapclient.com/xml/soapresponder.wsdl
```

- Permite conocer información sobre el servicio web
  - Esquema XML
  - Tipos
  - Operaciones
    - `Method1(bstrParam1: xsd:string, bstrParam2: xsd:string) ->`
      `bstrReturn: xsd:string`

soapresponder.wsdl

```
▼<definitions xmlns:tns="http://www.SoapClient.com/xml/SoapResponder.wsdl" xmlns:xsd1="h
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsd
 targetNamespace="http://www.SoapClient.com/xml/SoapResponder.wsdl">
  ▼<types>
     <schema xmlns="http://www.w3.org/1999/XMLSchema" targetNamespace="http://www.SoapCli
   </types>
  ▼<message name="Method1">
     <part name="bstrParam1" type="xsd:string"/>
     <part name="bstrParam2" type="xsd:string"/>
   </message>
  ▼<message name="Method1Response">
     <part name="bstrReturn" type="xsd:string"/>
   </message>
  ▼<portType name="SoapResponderPortType">
    ▼<operation name="Method1" parameterOrder="bstrparam1 bstrparam2 return">
       <input message="tns:Method1"/>
       <output message="tns:Method1Response"/>
     </operation>
   </portType>
  ▼<binding name="SoapResponderBinding" type="tns:SoapResponderPortType">
```

Ejemplo básico: servicio web de Eco

ws-eco.py

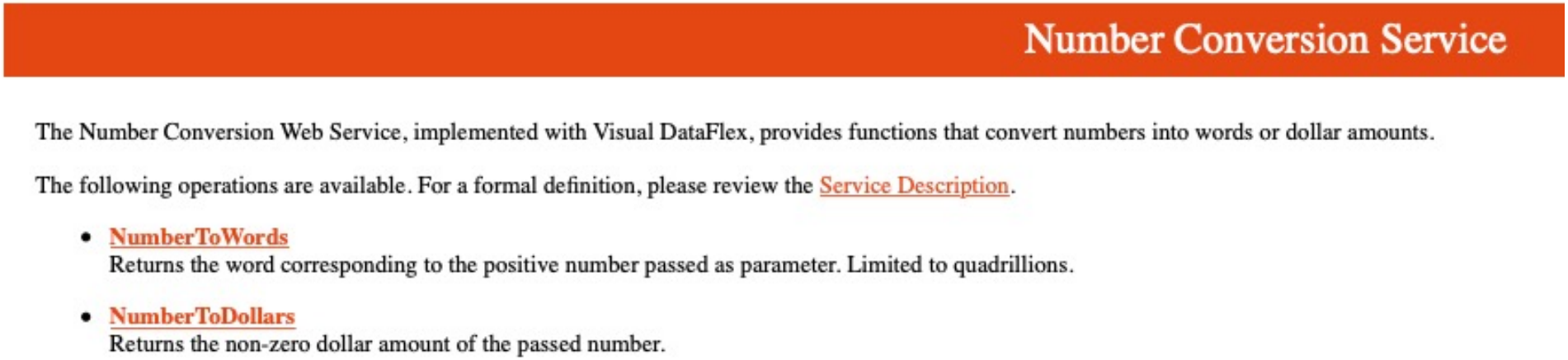
```
import zeep

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

Servicio web de conversión de números a palabras

- https://www.dataaccess.com/webservicesserver/numberconversion.wso



Number Conversion Service

The Number Conversion Web Service, implemented with Visual DataFlex, provides functions that convert numbers into words or dollar amounts.

The following operations are available. For a formal definition, please review the Service Description.

- **NumberToWords**
  Returns the word corresponding to the positive number passed as parameter. Limited to quadrillions.
- **NumberToDollars**
  Returns the non-zero dollar amount of the passed number.

- WSDL:
  - http://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL

Servicio web de conversión de números a palabras

```
python3 -mzeep
  https://www.dataaccess.com/webservicesserver/number
  conversion.wso?WSDL
```

- Operaciones
  - `NumberToDollars(dNum: xsd:decimal) ->`
    `NumberToDollarsResult: xsd:string`
  - `NumberToWords(ubiNum: xsd:unsignedLong) ->`
    `NumberToWordsResult: xsd:string`

Ejemplo: Servicio web de conversión de números a palabras

ws-ntow.py

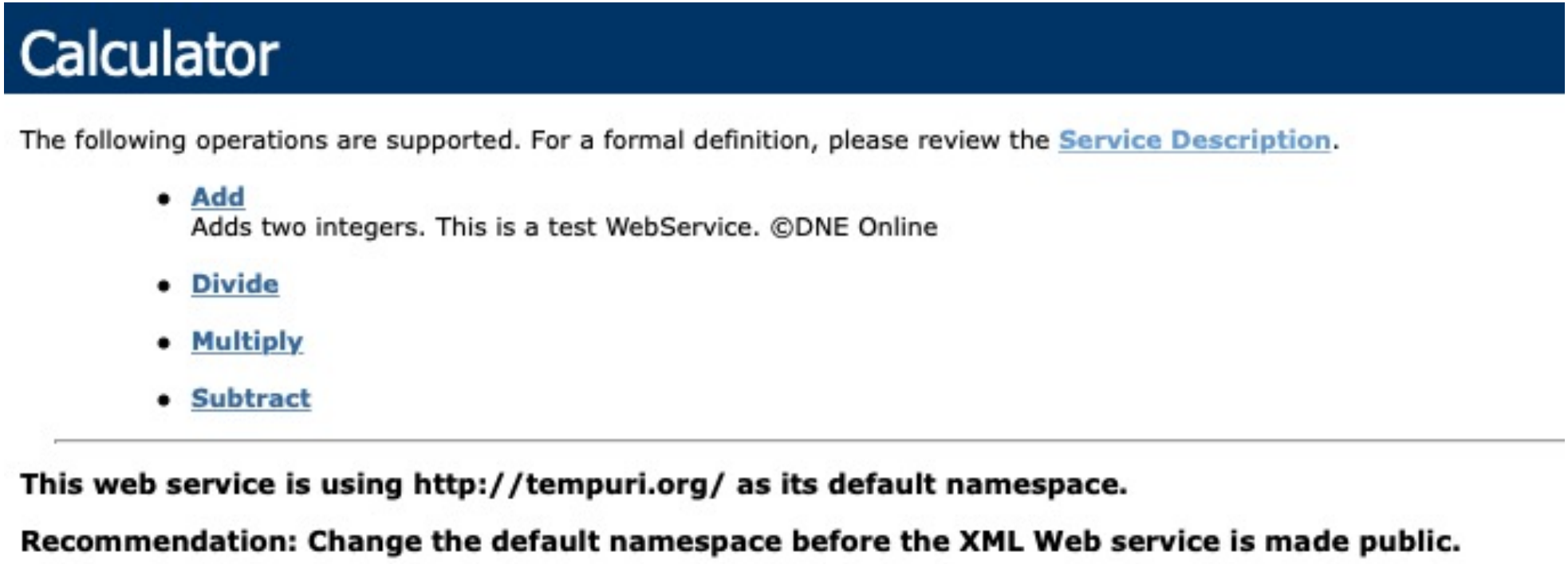
```
import zeep

wsdl =
'https://www.dataaccess.com/webservicesserver/numberco
nversion.wso?WSDL'
client = zeep.Client(wsdl=wsdl)
print(client.service.NumberToWords(427)
```

- Salida:
  - four hundred and twenty seven

Ejemplo: Servicio web calculadora

- WSDL:
  - http://www.dneonline.com/calculator.asmx?WSDL
- Servicios en http://www.dneonline.com/calculator.asmx:



Ejemplo: Servicio web calculadora

```
python3 -mzeep "http://www.dneonline.com/calculator.asmx?WSDL"
```

- Operaciones:

```
Operations:
   Add(intA: xsd:int, intB: xsd:int) -> AddResult: xsd:int
   Divide(intA: xsd:int, intB: xsd:int) -> DivideResult: xsd:int
   Multiply(intA: xsd:int, intB: xsd:int) -> MultiplyResult: xsd:int
   Subtract(intA: xsd:int, intB: xsd:int) -> SubtractResult: xsd:int
```

Ejemplo: Servicio web calculadora

ws-calc.py

```
import zeep

def main():
    wsdl_url = "http://www.dneonline.com/calculator.asmx?WSDL"
    soap = zeep.Client(wsdl=wsdl_url)
    result = soap.service.Add(5, 5)
    print(result)
    result = soap.service.Multiply(5, 5)
    print(result)


if __name__ == '__main__':
    main()
```

Spyne

- Modelo basado en SOAP para Python para creación de servicios.
- Despliegue del servidor.
- Generador de WDSL.

Instalación

- Instalación en el sistema:
  - pip install spyne

- Instalación para un usuario:
  - pip install spyne --user

Web service Calculadora (servidor)

ws-calc-service.py

```
import time

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
```

Web service Calculadora (servidor)

ws-calc-service.py

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

application = WsgiApplication(application)

if __name__ == '__main__':
    import logging

    from wsgiref.simple_server import make_server
    logging.basicConfig(level=logging.DEBUG)
    logging.getLogger('spyne.protocol.xml').setLevel(logging.DEBUG)
    logging.info("listening to http://127.0.0.1:8000")
    logging.info("wsdl is at: http://localhost:8000/?wsdl")
    server = make_server('127.0.0.1', 8000, application)
    server.serve_forever()
```

Web service Calculadora (cliente)

```
python3 ws-calc-service.py
```

```
python3 -mzeep http://localhost:8000/?wsdl
```

- Operaciones:

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

Web service Calculadora (cliente)

ws-calc-service.py

```
import zeep

wsdl = "http://localhost:8000/?wsdl"
client = zeep.Client(wsdl=wsdl)
print(client.service.add(5, 2))
client = zeep.Client(wsdl=wsdl)
print(client.service.sub(5, 2))
```

Servicios Web Rest (REpresentational State Transfer)

- Servicio web basado en el concepto de recurso. Recurso es cualquier element con una URI
- No es un estándar, es un estilo de arquitectura
- Características:
  - Interfaces construidas sobre HTTP:
    - GET: obtener un recurso
    - DELETE: borrar un recurso
    - POST: para actualizar o crear recursos
    - PUT: para crear recursos
- Mensajes en HTTP, XML, json…
- Mensajes simples codificados en las URL

Características de REST

- Estilo cliente-servidor
- Sin estado
- Todos los servicios accesibles a través de los métodos GET, POST, PUT y DELETE
- Sencillez de invocación respecto a SOAP y más ligeros
- Los recursos tienen nombres basados en URL
- Ejemplo:
  - Obtener los datos de un componente:
    - http://host/componente/23456

gSOAP

- **gSOAP**: Generator Tools for Coding SOAP/XML Web Services in C and C++ (Robert van Engelen, University of Florida)

- Entorno de desarrollo independiente de la plataforma para el desarrollo servicios web en C/C++

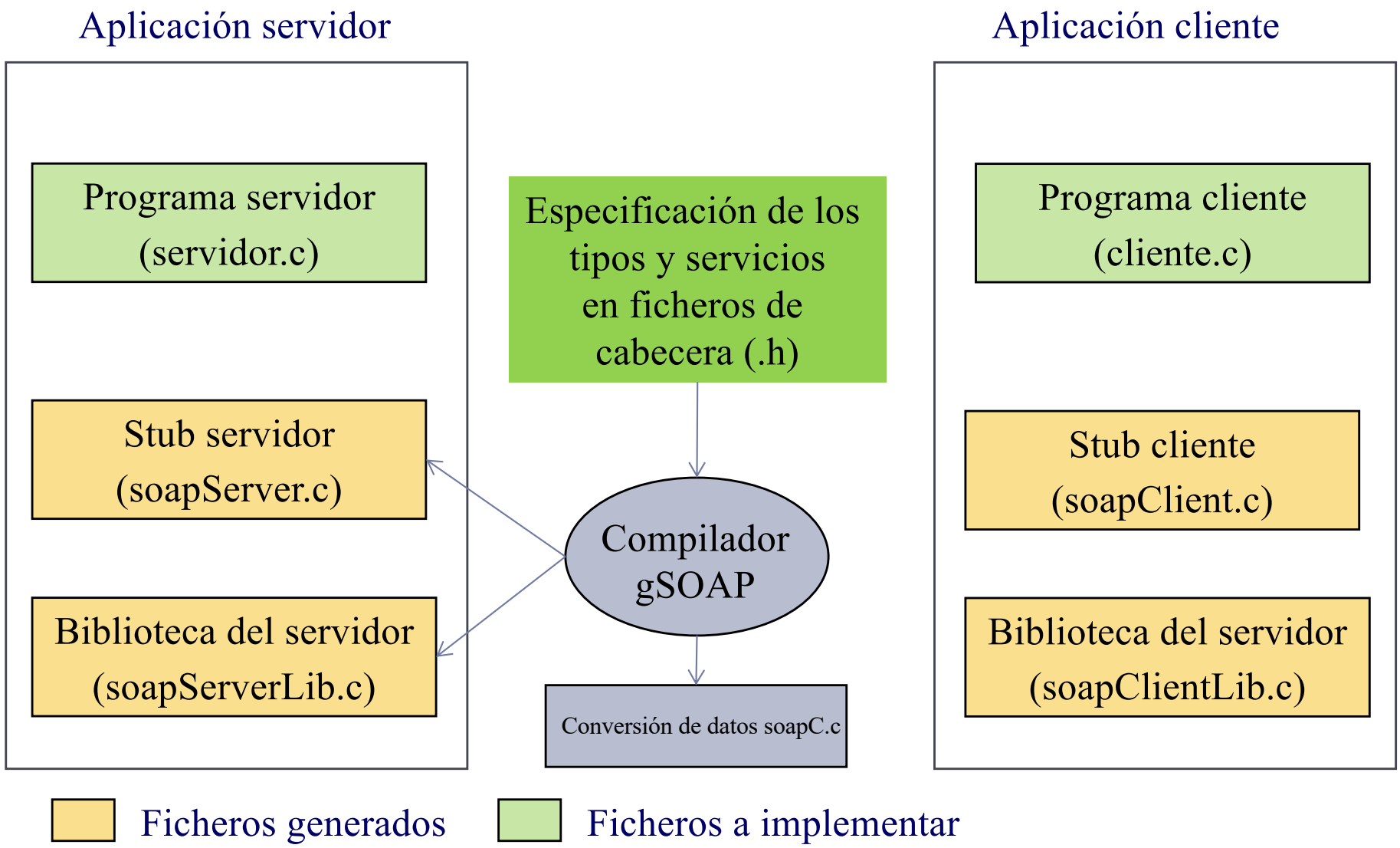
- gSOAP proporciona un **compilador** que genera código del stub y los esqueletos de las rutinas para integrar las aplicaciones escritas en C y C++ con los servicos web SOAP/XML

- http://www.cs.fsu.edu/~engelen/soap.html

Carcterísticas de gSOAP

- Conversión SOAP←→C/C++
- **Independiente** de la plataforma
- Inicialmente pensado para C++, soporta el desarrollo de aplicaciones escritas en C
- Soporta el desarrollo de aplicaciones stand-alone multithread
- Tolerancia a fallos
- Versión de gSOAP sobre **UDP**
- Soporta las versiones **IPv4** e **IPv6**
- Soporta **autenticación** HTTP
- Gestión de memoria
- Marshalling/Unmarshalling de datos

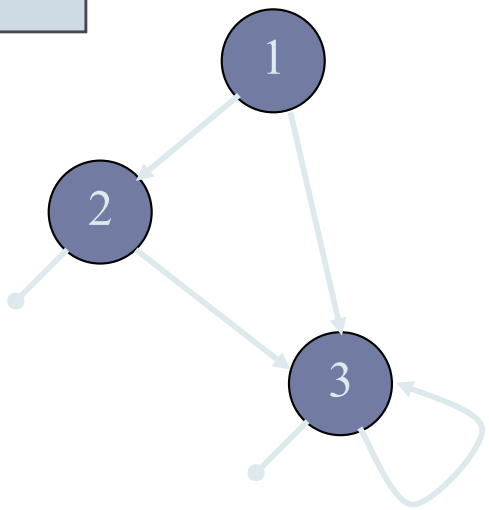
Desarrollo de una aplicación



Marshalling/unmarshalling

- gSOAP se ocupa del proceso de serialización del código C/C++ a XML (**marshalling**) y de deserialización de XML a C/C++ (**unmarshalling**)

```
struct BG
{ int val;
  struct BG *left;
  struct BG *right;
};
```

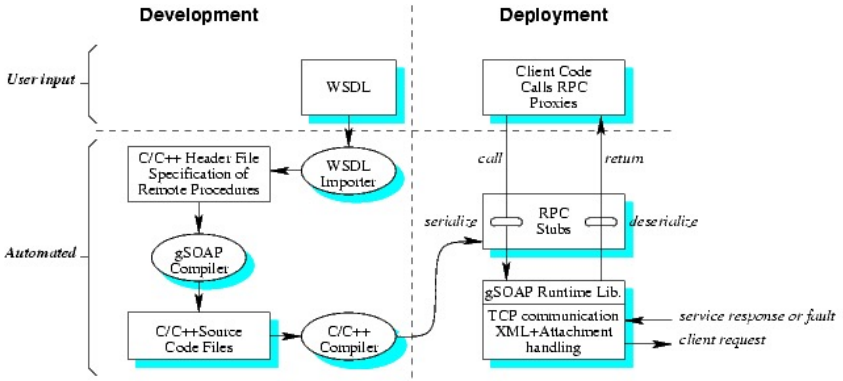



```
<BG>
 <val>1</val>
 <left>
  <val>2</val>
  <right href="#X"/>
 </left>
 <right href="#X"/>
</BG>
<id id="X">
 <val>3</val>
 <right href="#X"/>
</id>
```

Herramientas proporcionadas por gSOAP

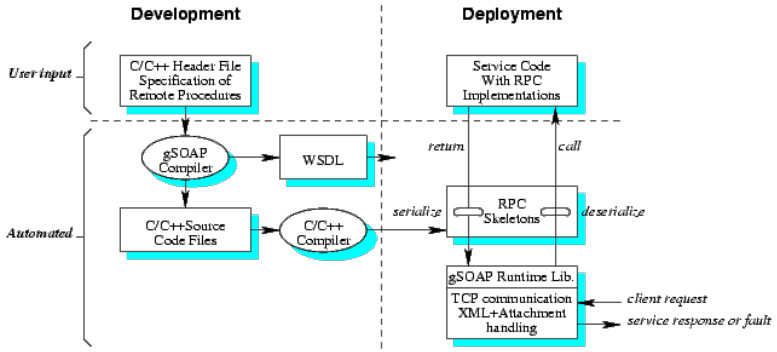
Wsdl2h

wsdl2h es un parseador de código WSDL y esquemas de XML que genera un fichero de cabecera con los servicios web y los tipos de datos en C/C++ usados por esos servicios



soapcpp2

soapcpp2 es un compilador que a partir de un fichero de cabecera de C/C++ genera código de los stubs del cliente, los esqueletos de los servicios web implementados en el servidor y los ficheros para la serialización/ deserialización de los datos



http://www.cs.fsu.edu/~engelen/soap.html

Ejemplo de uso

- wsdl2h

```
wsdl2h  -o XMethodsQuery.h http://www.xmethods.net/wsdl/query.wsdl
```

  - donde
    - XmethodsQuery.h — Fichero generado por wsdl2h
    - http://www.xmethods.net/wsdl/query.wsdl — Fichero de entrada en formato WSDL
  - La opción –o nombre_fichero.h especifica el nombre del fichero de salida

- soapcpp2

```
soapcpp2 -c XMethodsQuery.h
```

  - donde
    - XmethodsQuery.h — Fichero de cabecera de entrada
  - La opción –c especifica que se generará código en el lenguaje C

Especificación de servicios web en gSOAP

- Un fichero de cabecera de definición de servicios web puede contener un **conjunto de directivas** que especifican las propiedades del servicio
  - Especificar nombre de servicio

    // gsoap namespace-prefix service name: myservice-name
  - Especificar una acción SOAPAction para un método

    // gsoap namespace-prefix service method-action: method-name-action
  - Especificar la localización del servicio mediante una URL

    // gsoap namespace-prefix service location: url

    // gsoap namespace-prefix service port:url
  - Especificar la URI del espacio de nombres de un servicio web

    // gsoap namespace-prefix service namespace: namespace-URI
  - Especificar la URI del esquema del espacio de nombres de un servicio web

    // gsoap namespace-prefix squema namespace: namespace-URI

Ejemplo: calculadora

```
// contents of file calc.h
// gsoap ns service name: calculator
// gsoap ns service style: rpc
// gsoap ns service encoding: encoded
// gsoap ns service port: http://mydomain/path/calculator.cgi
// gsoap ns service namespace: urn:calculator

int ns__add(double a, double b, double *result);
int ns__sub(double a, double b, double *result);
```

Invocación de servicios web en gSOAP

- Los servicios web se especifican en un fichero de cabecera (.h) como prototipos de funciones de C++

```
[int] [namespace_prefix__]method_name([inparam1,
      inparam2], &outparam)
```

- Se usan las siguientes convenciones:
  - **A cada nombre de función del servicio web** debe anteponerse un prefijo del espacio de nombres seguido de dos guiones bajos "__"
  - **Un servicio web siempre devuelve un valor de tipo int** que se usa para conocer si el servicio web finalizó con éxito o error
  - **Un servicio web puede aceptar de 0 a n argumentos de entrada**. Todos los argumentos de entrada deben pasarse por valor o usar una referencia de C++ o un puntero de C
  - **Sólo habrá un argumento de salida para un servicio web**, que se especifica como último argumento del servicio web. El argumento de salida debe siempre ser una referencia o un puntero de C

El programa cliente (I)

- Antes de invocar uno o varios servicios web el cliente debe crear un **entorno de ejecución** de gSOAP:
  - Inicializar un entorno de ejecución estáticamente. **Obligatorio**

```
int soap_init(struct soap *soap)
```

  - Asignar e inicializar un entorno de ejecución. Devuelve un puntero a un entorno de ejecución.

```
struct  soap* soap_new()
```

  - Copia el contenido de un entorno de ejecución a otro, de manera que ambos entornos no compartan datos

```
struct  soap* soap_copy(struct soap* soap)
```

El programa cliente (II)

- Un programa cliente podrá **invocar** los servicios web usando el siguiente prototipo de función:

```
int soap_call_[namespace_prefix__]method_name(struct soap
*soap, char *URL, char *action, [inparam1,
              inparam2, …], &outparam)
```

- Donde
  - namespace_prefix__ Prefijo del espacio de nombres especificado en el archivo de cabecera
  - method_name Nombre del servicio web especificado en el archivo de cabecera
  - soap Entorno de ejecución de Gsoap del cliente
  - URL Localización del servicio web
  - action Una acción de las especificadas en el API de SOAP. Si opcional → “”
  - inparam1, inparam2,.. La lista de argumentos de entrada: tipo name
  - outparam El argumento de salida del servicio web

El programa cliente (III)

- Después de invocar uno o varios servicios web el cliente debe eliminar el **entorno de ejecución**
  - Borrar instancias de clases (sólo para C++)

```
int soap_destroy(struct soap *soap)
```

  - Liberar los recursos de un entorno de ejecución creado estáticamente

```
int soap_end()
```

  - Liberar los recursos de un entorno de ejecución creado dinámicamente

```
int soap_free()
```

  - Eliminar el entorno de ejecución: cierra la comunicación y libera los recursos

```
int soap_done()
```

El programa servidor (I)

- El programa servidor deberá crear un **entorno de ejecución** antes de poder proporcionar los servicios web al cliente
  - Usa las mismas llamadas que el programa cliente
- Además, el servidor debe:
  - Implementar el bucle principal que atiende las peticiones de los clientes. Los servicios web pueden proporcionarse de dos maneras:
    - Servidor usando CGI (common Gateway Interface)
    - Servidor stand-alone
  - Implementar las funciones (servicios web) especificados en el fichero de cabecera de entrada
  - Finalmente, destruir el entorno de ejecución cuando quiera dejar de proporcionar los servicios a los clientes
    - Usa las mismas llamadas que el programa cliente

El programa servidor (II)

- El servidor utiliza el siguiente prototipo de función para activar el servicio web correspondiente al solicitado por el cliente

```
int soap_serve (struct soap *soap)
```

- donde

  soap Entorno de ejecución de gSOAP en el servidor

  - Implementación del servicio web:
    - Si el servicio web solicitado es un CGI, entonces el servidor únicamente deberá invocar la rutina soap_serve
    - Si el servicio web solicitado es una función implementada en un servidor stand-alone, el servidor deberá invocar funciones análogas a las usadas en sockets

Ejemplo: programa servidor basado en CGI

```
#include "soapH.h"
#include "ns.nsmap"
int main(){
   /* Entorno de ejecución del servidor */
   struct soap soap;
   /* Inicializar un entorno de ejecución */
   soap_init(&soap);
   /* Activar un servicio web */
   soap_serve(&soap);
   /* Liberar recursos y destruir un entorno de ejecución */
   soap_end(&soap);
   soap_done(&soap);
   exit(0);
}
/* Implementación de los servicios web */
int mi_funcion(input1,input2,output){
   /* Algún tratamiento */
   return SOAP_OK;
}
```

El programa servidor (III)

- El servidor puede proporcionar los servicios web como **stand-alone** usando el protocolo HTTP y cualquier puerto
- Para ello deberá usar las siguientes funciones del API de gSOAP (funciones análogas a las de sockets stream)

  - Devuelve el socket primario del servidor. Si host es NULL se toma la máquina en que se ejecuta el servidor

```
int soap_bind(struct soap *soap, char *host, int
              port, int backlog)
```

  - Devuelve el socket secundario correspondiente a la conexión entrante

```
int soap_accept(struct soap *soap)
```

Ejemplo: programa servidor *stand-alone* (I)

```
#include "soapH.h"
#include "ns.nsmap"
int main(){
   int m,s;
   struct soap soap;     /* Entorno de ejecución del servidor */
   soap_init(&soap);     /* Inicializar un entorno de ejecución */
   if (m=soap_bind(&soap, "machine.cs.fsu.edu", 18083, 100))<0){
         soap_print_fault(&soap,stderr);
         exit(-1);
   }
   ….
```

Ejemplo: programa servidor *stand-alone* (II)

```
   ….
   for (int i=1;;i++){
         if ((s=soap_accept(&soap))<0){
                  soap_print_fault(&soap,stderr);
                  exit(-1);
         }
         if (soap_serve(&soap)!=SOAP_OK){   /* Activar un servicio web */
                  soap_print_fault(&soap,stderr);
                  exit(-1);
         }
         soap_end(&soap);   /* limpiar y cerrar socket secundario */
   }
   soap_done(&soap);         /* Destruir un entorno de ejecución */
   exit(0);
}
/* Implementación de los servicios web */
```

Ejemplo:
programa servidor *stand-alone multithread* (I)

```
#include "soapH.h"
#include "ns.nsmap"
int main(){
   int m,s;
   struct soap soap;     /* Entorno de ejecución del servidor */
   struct soap *soap2;
   soap_init(&soap);     /* Inicializar un entorno de ejecución */
   if (m=soap_bind(&soap, "machine.cs.fsu.edu", 18083, 100))<0){
        soap_print_fault(&soap,stderr);
        exit(-1);
   }
   ….
```

Ejemplo:
programa servidor *stand-alone multithread* (II)

```
….
for (int i=1;;i++){
        if ((s=soap_accept(&soap))<0){
                soap_print_fault(&soap,stderr);
                exit(-1);
        }
        soap2 = soap_copy(&soap); // make a safe copy
        if (!soap2)
          break;
        pthread_create(&tid, NULL, (void*(*)(void*))process_request, (void*)soap2);
    }
    soap_done(&soap); // detach soap struct
}
void *process_request(void *soap)
{
    pthread_detach(pthread_self());
    soap_serve((struct soap*)soap);
    soap_destroy((struct soap*)soap);          // dealloc C++ data
    soap_end((struct soap*)soap);              // dealloc data and clean up
    soap_done((struct soap*)soap);             // detach soap struct
    return NULL;
}
```

Más ejemplos

- WSDL:
  - http://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL
  - Accesible desde http://www.xmethods.com/

- Dos métodos:

  string **NumberToWords**(unsignedLong ubiNum)
  // Returns the word corresponding to the positive number passed as parameter. Limited to quadrillions.

  string **NumberToDollars**(decimal dNum)
  // Returns the non-zero dollar amount of the passed number

Elementos del WSDL

```
<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" />

<message name="NumberToWordsSoapRequest">
<part name="parameters" element="tns:NumberToWords" />
</message>

<message name="NumberToWordsSoapResponse">
<part name="parameters"
 element="tns:NumberToWordsResponse" />
</message>
```

Elementos del WSDL

```
<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>
```

Generación de un cliente

- **Paso 1**: Obtener el archivo de cabecera a partir del WSDL:
  - wsdl2h -c -o conversions.h http://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL

- **Paso 2:** Generación de los stub y skeletons a partir del archivo de cabecera
  - soapcpp2 -C -c conversions.h

- **Paso 3:** Implementación del cliente

Ejemplo: implementación del programa cliente

```
#include "soapH.h" // obtain the generated stub
#include "NumberConversionSoapBinding.nsmap"

main(int argc, char **argv)
{
   int err;
   struct _ns1__NumberToWords  arg1;
   struct _ns1__NumberToWordsResponse arg2;

   struct soap *soap = soap_new();

   arg1.ubiNum = atoi(argv[1]); // argumento

   err = soap_call___ns1__NumberToWords (soap, NULL, NULL, &arg1, &arg2);
  if (err == SOAP_OK)
      printf("Resultado = %s\n", arg2.NumberToWordsResult);
   else // an error occurred
      soap_print_fault(soap, stderr);
}
```

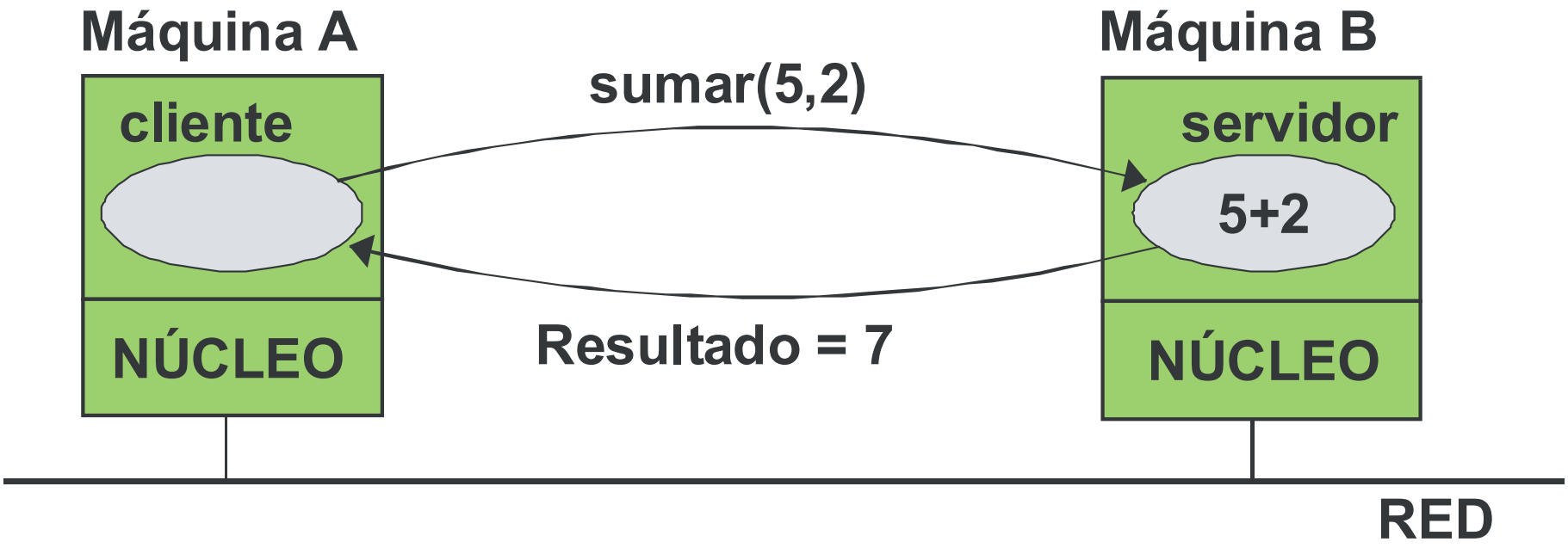
Compilación y ejecución

- Se compilan los siguientes ficheros fuente y se obtiene el programa client
  - gcc -o client client.c soapC.c soapClient.c libgsoap.a

- Ejecución:
  - `./client 34`

```
Resultado = thirty four
```

Ejemplo VIII:
Programar un servidor y cliente en TCP



Ejemplo: programa calculadora

- **Paso 1.** Fichero de cabecera **calc.h**

```
// gsoap ns service location: http://localhost:9000

int ns__sumar(int a, int b, int *result);
int ns__restar(int a, int b, int *result);
```

- **Paso 2**: generación de stubs y skeletons:

```
soapcpp2 -c calculadora.h
```

Paso 3:
Desarrollo del servidor (I)

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

int main(int argc, char **argv){
  int m, s; /* master and slave sockets */
  struct soap soap;
  soap_init(&soap);
  if (argc < 2)
    soap_serve(&soap);  /* serve as CGI application */
  else
  { m = soap_bind(&soap, NULL, atoi(argv[1]), 100);
    if (m < 0)
    {
        soap_print_fault(&soap, stderr);
        exit(-1);
    }
    fprintf(stderr, "Socket connection successful: master socket
                                = %d\n", m);
```

Paso 3:
Desarrollo del servidor (II)

```
  for ( ; ; )
  {
       s = soap_accept(&soap);
      fprintf(stderr, "Socket connection successful:
                      slave socket = %d\n", s);

       if (s < 0)
       {
              soap_print_fault(&soap, stderr);
              exit(-1);
       }

      soap_serve(&soap);
      soap_end(&soap);
  } /* end for */
  soap_done(&soap);
  } /* end else */
  return 0;
}
```

Paso 3:
Desarrollo del servidor (III)

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

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

Paso 4:
desarrollo del cliente (I)

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

// const char server[] =
// "http://websrv.cs.fsu.edu/~engelen/calcserver.cgi";
// const char server[] = "http://localhost:9000";

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

  if (argc != 2) {
    printf("Uso: calcClient   http://server:port\n");
    exit(0);
  }

  soap_init(&soap);
```

Paso 4:
desarrollo del cliente (II)

```
 server = argv[1];
 a = 5;
 b = 7;

 soap_call_ns__sumar (&soap, server, "", a, b, &res);

 if (soap.error)
  {
        soap_print_fault(&soap, stderr);
        exit(1);
  }
  printf("Resultado = %d \n", res);

//  soap_destroy(&soap)     solo para C++
  soap_end(&soap);
  soap_done(&soap);
  return 0;
}
```

Paso 5: compilación y ejecución

- Cliente:
  - gcc client.c soapC.c soapClient.c libgsoap.a -o client
- Servidor:
  - gcc server.c soapC.c soapServer.c libgsoap.a -o server

- >./server 9000

- > ./client http://localhost:9000