Servicios Web

Sesión 1: Introducción e invocación de Servicios Web

Puntos a tratar

- ¿Qué es un Servicio Web?
- Arquitectura de los Servicios Web
- Seguridad
- Tecnologías básicas
- Tecnologías J2EE para Servicios Web
- Tipos de clientes
- Creación de un stub estático
- Interfaz de Invocación Dinámica (DII)

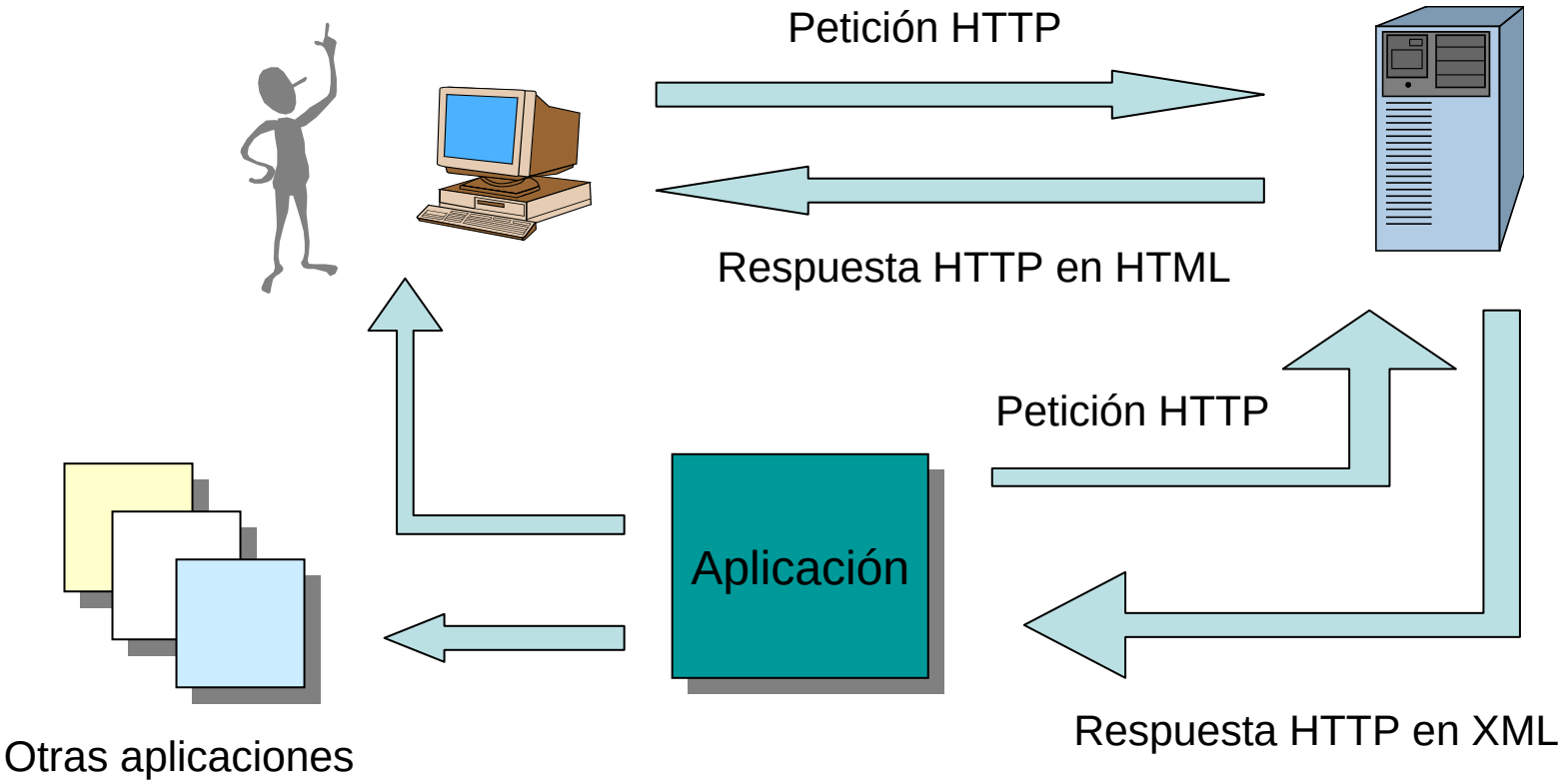
Componentes software

- El diseño del software tiende a ser cada vez más modular
  - Aplicaciones compuestas por componentes reutilizables
    P.ej. Objetos CORBA o EJBs
  - Estos componentes pueden encontrarse distribuidos
- Servicio
  - Unidad funcional de software
  - Ofrece una determinada interfaz y cumple ciertos requisitos
  - Deberá poder ser integrado en la aplicación y combinado con otros servicios de forma independiente.

Descripción de Servicio Web

- Un Servicio Web es un servicio al que se podrá acceder mediante protocolos Web estándar
  - Los mensajes para invocar el servicio se codifican en XML
  - Estos mensajes se pueden transportar utilizando HTTP
- Normalmente constará de una interfaz (conjunto de métodos) que podremos invocar de forma remota desde cualquier lugar de la red
  - Nos permiten crear aplicaciones distribuidas en Internet
- Son independientes de la plataforma y del lenguaje de programación en el que estén implementados

Web “para humanos” vs. “para máquinas”



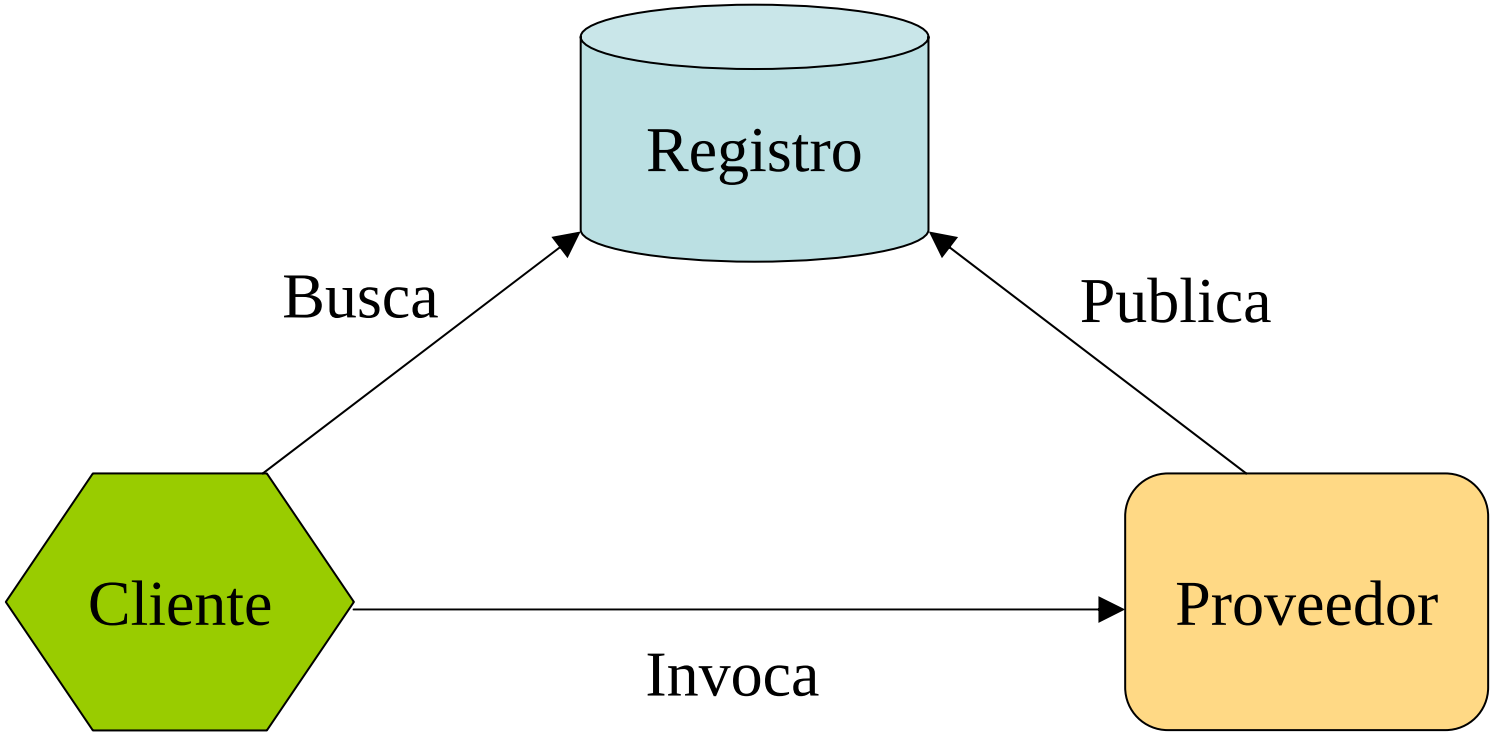
Características de los servicios

- Deben ser accesibles a través de la Web
  - Debe utilizar protocolos de transporte estándares como HTTP y codificar los mensajes en un lenguaje estándar (XML).
- Deben describirse a si mismos
  - De esta forma una aplicación podrá conocer cuál es la interfaz del servicio, y podrá integrarlo y utilizarlo de forma automática.
- Deben ser localizables
  - Debe existir algún mecanismo de localizar un servicio que realice una determinada función, sin tenerlo que conocer previamente el usuario.

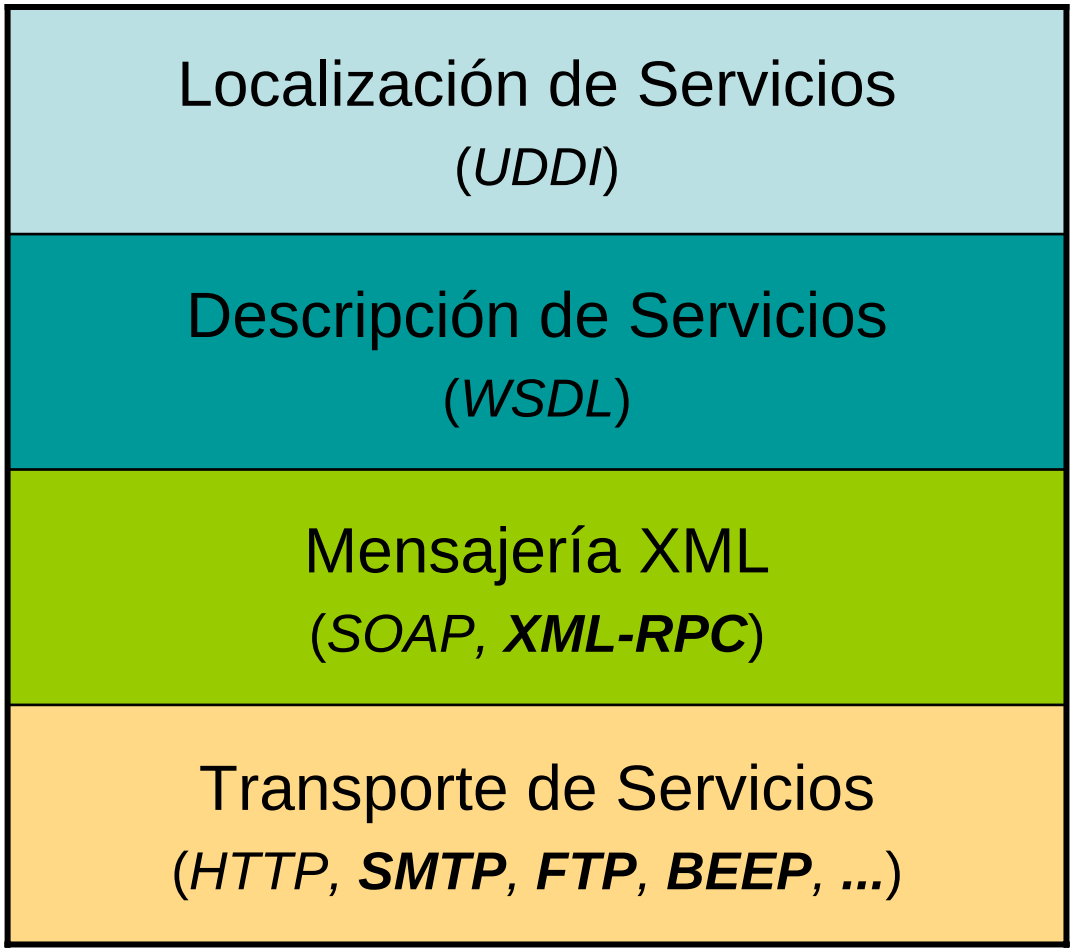
Arquitecturas Orientada a Servicios (SOA)

- Desarrollo de servicios altamente reutilizables
  - Interfaz estándar bien definida
  - Sin estado
  - No deben depender del estado de otros componentes
- Orquestación de servicios
  - Combinar servicios para construir aplicaciones
  - Se pueden formar diferentes flujos para implementar los procesos de negocio
- Los servicios pueden ser de diferentes tipos
  - Servicios Web, JMS, etc.

Componentes de una SOA



Capas de los Servicios Web



Confidencialidad

- La información contenida en los mensajes SOAP puede ser confidencial
- Solución:
  - Como los mensajes se envían por HTTP, podemos encriptarlos con SSL
- Problema:
  - Si el mensaje debe atravesar una cadena de servicios, debe ser desencriptado dentro de cada uno de ellos

    Los datos estarán inseguros dentro de cada nodo
  - Solución:

    Desencriptar sólo la parte concerniente a cada nodo

Autentificación

- Podemos necesitar identificar al usuario
  - Para prestarle un servicio personalizado
  - Para comprobar si tiene permiso para usar el servicio
  - Etc...
- Solución:
  - Utilizar autentificación HTTP
- Problema:
  - Si utilizamos servicios de distintos servidores, tendremos que autentificarnos para cada uno por separado
  - Solución:

    Crear contexto compartido donde puedan consultar información sobre la autentificación (P.ej. MS Passport, Liberty Project)

Seguridad en la red

- Se invocan procedimientos remotos mediante HTTP
  - Protocolo diseñado para extracción de documentos
- Los *firewalls* permiten el paso de este tipo de peticiones
  - No se puede cortar el acceso a estos puertos
- Problema de seguridad
  - Debemos tener en cuenta que cualquiera va a poder utilizar estos servicios

SOAP

- Protocolo derivado de XML
- Se usa para intercambiar información
- Dos tipos:
  - Mensajes orientados al documento
    - Cualquier tipo de contenido
  - Mensajes orientados a RPC
    - Tipo más concreto que el anterior
    - Nos permite realizar llamadas a procedimientos remotos
      - La petición contiene el método a llamar y los parámetros
      - La respuesta contiene los resultados devueltos
- Nos centraremos en el segundo tipo

Elementos de SOAP

- Sobre SOAP (*Envelope*). Contiene:
  - Descripción del mensaje (destinatario, forma de procesarlo, definiciones de tipos)
  - Cabecera (opcional) y cuerpo SOAP
- Cabecera SOAP (*Header*). Contiene:
  - Información sobre el mensaje (obligatorio, actores, etc)
- Cuerpo SOAP (*Body*). Contiene:
  - Mensaje (en caso de RPC la forma del mensaje se define por convención)
  - Error (opcional)
- Error SOAP (*Fault*)
  - Indica en la respuesta que ha habido un error en el procesamiento de la petición

Elementos de SwA

- Con SOAP podemos intercambiar cualquier documento XML, pero no otro tipo
  - Por ejemplo, una imagen.
- SwA (SOAP with Attachment) nos permite añadir datos que no sean XML al mensaje
- Parte adjunta (*Attachment*)
  - Contiene los datos no XML

Ejemplo SOAP

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
  <ns:getTemperatura xmlns:ns="http://j2ee.ua.es/ns">
    <area>Alicante</area>
  </ns:getTemperatura>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

WSDL

- Lenguaje derivado de XML
- Describe la interfaz de los Servicios Web
  - Operaciones disponibles
  - Parámetros de las operaciones
  - Resultados devueltos
  - Tipos de datos de estos parámetros y resultados
- Además contiene la dirección del *endpoint*
  - URL a la que hay que conectarse para acceder al servicio
- Nos permite integrar un servicio automáticamente en nuestra aplicación, o que otros usuarios utilicen los servicios que hayamos desarrollado nosotros

Elementos WSDL

Elementos de WSDL lógicos

- El elemento raíz del documento es `definitions`, contiene:
  - `types`: Tipos de datos que se intercambian
  - `message`: Mensajes que se intercambian durante la invocación de las operaciones. Cada operación tendrá un mensaje de entrada (petición) y uno de salida (respuesta)
  - `portType`: Define las operaciones que ofrece el servicio. Cada una tendrá un mensaje de entrada y salida de los anteriores

Elementos de WSDL físicos

- `binding`: Indica protocolo y formato para los mensajes anteriores. El formato puede ser:
  - Orientado al documento
  - Orientado a RPC
- `service`: Define el servicio mediante una colección de puertos a los que acceder.
  - Cada puerto tendrá una URL para acceder al *endpoint*.
  - Además contiene documentación en lenguaje natural sobre el servicio.

UDDI

- UDDI nos permite localizar Servicios Web
- Define la especificación para construir un directorio distribuido de Servicios Web
  - Se registran en XML
- Define una API para acceder a este registro
  - Buscar servicios
  - Publicar servicios
- La interfaz de UDDI está basada en SOAP
  - Se utilizan mensajes SOAP para buscar o publicar servicios

Tecnologías de segunda generación

- Una vez establecidas las tecnologías básicas, aparecen extensiones:
  - *WS-Policy* y *WS-PolicyAttachment*

    Describir funcionalidades que no podemos especificar con WSDL.
  - *WS-Security*

    Seguridad a nivel de mensaje.
  - *WS-Addressing* y *WS-ReliableMessaging*

    Servicios Web asíncronos.
  - *WS-Coordination* o *BPEL*

    Orquestar servicios web.

JAXP

- Permite procesar documentos XML en Java
- Tiene en cuenta espacios de nombres
- Soporta XSLT
  - Podemos transformar XML a otros formatos
- Librería para tratar XML genérico
  - Otras librerías se apoyan en esta para procesar tipos concretos de lenguajes derivados de XML

    SOAP

    WSDL

    UDDI

JAXM

- Mensajería XML orientada al documento
  - Trabaja con mensajes SOAP y SwA
- Nos permite
  - Extraer el contenido de los mensajes XML recibidos
  - Crear y enviar mensajes XML
    - Síncrona (petición-respuesta)
    - Asíncrona (envío sin esperar respuesta)
- Se divide en dos APIs
  - SAAJ: API independiente y suficiente para:
    - Crear mensajes SOAP y extraer información de ellos
    - Envío síncrono de mensajes
  - JAXM: API dependiente de SAAJ. Incorpora:
    - Envío asíncrono de mensajes

JAX-RPC / JAX-WS

- Infraestructura para hacer RPC mediante XML
  - Utiliza mensajes SOAP orientados a RPC
- Depende de SAAJ, pero no de JAXM
  - SAAJ se encarga de
    - Construir y enviar los mensajes
    - Recibir y analizar los mensajes
- Nos permitirá:
  - Invocar Servicios Web de tipo RPC
  - Crear nuestros propios Servicios Web RPC
    - A partir de clases Java que implementan su funcionalidad
- A partir de la versión 2.0 pasa a llamarse JAX-WS
  - Utiliza anotaciones para definir los servicios

JAXR

- Permite acceder a registros XML
  - UDDI
  - ebXML
- Utiliza una API estándar Java
  - Se accede de la misma forma a cualquier tipo de registro
- Permite
  - Consultar el registro
  - Publicar servicios en el registro
  - Eliminar o modificar los servicios publicados

JAXB

- Permite asociar esquemas XML a clase Java
- Convierte los tipos de datos utilizados en el servicio:
  - Unmarshalling

    XML → Objeto Java
  - Marshalling

    Objecto Java → XML

- Otras APIs
  - Java API for WSDL (WSDL4J)

    Analiza y construye documentos WSDL

    Podemos consultar la interfaz de un servicio a partir de su documento WSDL

    Permitirá integrar servicios en tiempo de ejecución

Interoperabilidad de servicios

- Podremos crear clientes para utilizar cualquier servicio
  - Se invocan mediante protocolos Web estándar
  - Accederemos a cualquier servicio de la misma forma
  - No importa el lenguaje o plataforma del *endpoint*
- Las tecnologías Java de Servicios Web
  - Cumplen *WS-I Basic Profile* (BP)

    Estándar de interoperabilidad para Servicios Web

    Definido por *Web Services Interoperability Organization*

    Define una serie de reglas para aclarar ambigüedades

    Las especificaciones de SOAP, WSDL y UDDI no son claras

    El uso conjunto de estas tecnologías se especifica en BP
  - Serán interoperables con cualquier servicio BP

Tipos de acceso

- JAX-RPC/WS nos permite acceder de 2 formas:
  - Creación de un *stub* estático
    - Se genera una capa *stub* en tiempo de compilación
    - Esta capa se genera automáticamente mediante herramientas
    - El cliente accede a través del *stub* como si fuese a un objeto local
  - Interfaz de invocación dinámica (DII)
    - Se hacen llamadas de forma dinámica, sin *stub*
    - Se proporcionan los nombres de las operaciones a ejecutar mediante cadenas de texto a métodos genéricos de JAX-RPC
    - Se pierde transparencia

Librería JAX-RPC / JAX-WS

- Weblogic incluye varias implementaciones de JAX-RPC

`webserviceclient.jar`	Implementación de JAX-RPC
`webserviceclient+ssl.jar`	Implementación para utilizar SSL

  - Se encuentran en `bea/weblogic92/server/lib`

- A partir de JDK 1.6 se incluye JAX-WS en Java SE
  - En versiones previas se puede incluir esta librería o JAX-RPC
  - JAX-WS no es compatible con servicios `rpc/encoded`

- Deberemos añadir a nuestro cliente la librería adecuada

Acceso mediante stub estático

- Es la forma más sencilla de acceder
  - Necesitamos una herramienta que genere el *stub* de forma automática
    P.ej. en el caso de Weblogic, utilizaremos la tarea `clientgen`
  - El *stub* implementará la misma interfaz que nuestro servicio
  - Utilizaremos el *stub* para acceder al servicio como si fuese un objeto local
  - Es totalmente transparente para nuestro cliente que se esté invocando un Servicio Web
  - No será necesario utilizar código JAX-RPC/WS en nuestro cliente, esta tarea la hace el *stub* generado

Generar el stub con Weblogic

- Generaremos el *stub* a partir del WSDL
  - Nos servirá para cualquier servicio descrito mediante un documento WSDL estándar
  - No importa la implementación del servicio
- Utilizamos la tarea `clientgen` de Weblogic

```
<clientgen
  wsdl="http://jtech.ua.es/HolaMundo/wsdl/HolaMundoSW.wsdl"
  packageName="es.ua.jtech.servcweb.hola.stub"
  destDir="src" />
```

Generar el cliente con JAX-WS

- Se utiliza la herramienta `wsimport`

```
wsimport -s src -d bin
    -p es.ua.jtech.servcweb.hola.stub
    http://jtech.ua.es/HolaMundo/wsdl/HolaMundoSW.wsdl
```

- También disponible como tarea de ant

```
<wsimport sourcedestdir="${src.home}"
    destdir="${bin.home}" package="${pkg.name}"
    wsdl="${wsdl.uri}" />
```

- Netbeans permite generar clientes para servicios de forma automática

Acceso al servicio

- Una vez obtenido el *stub* (`port`), accedemos al servicio como si fuese un objeto local

```
public class Main {
  public static void main(String[] args)
                          throws Exception {
    HolaMundoSWService service =
                         new HolaMundoSWService();
    HolaMundoSW port = service.getHolaMundoSW();
    System.out.println("Resultado: " +
                         port.saluda("Miguel"));
  }
}
```

Interfaz de invocación dinámica

- No se utiliza un *stub* para invocar las operaciones
  - Se invocan de forma dinámica
  - Nos permite invocar servicios que no conocemos en tiempo de compilación
- Utilizamos directamente la librería JAX-RPC/WS
  - Perdemos totalmente la transparencia
- JAX-RPC/WS proporciona métodos genéricos para invocar servicios
  - Indicamos el nombre de la operación mediante una cadena de texto
  - Indicamos los parámetros como un *array* de objetos

Con documento WSDL

```
// Obtenemos el servicio
ServiceFactory sf = ServiceFactory.newInstance();
Service serv = sf.createService(
  new URL(
   "http://localhost:7001/Conversion/Conversion?WSDL"),
   new QName("http://jtech.ua.es", "Conversion"));

// Creamos la llamada a la operacion
Call call = serv.createCall(
  new QName("http://jtech.ua.es","ConversionSoapPort"),
  new QName("http://jtech.ua.es","euro2ptas"));

// Invocamos la operacion
Integer result = (Integer) call.invoke(
                  new Object[] { new Double(30.0) });
```

Sin documento WSDL

- Podemos utilizar servicios sin proporcionar un documento WSDL

```
Service serv = sf.createService(
  new QName("http://jtech.ua.es", "Conversion"));
```

- Antes de invocar la operación se debe indicar la siguiente información:

```
call.setTargetEndpointAddress(endpointURL);

QName t_int = new
  QName("http://www.w3.org/2001/XMLSchema", "int");
call.setReturnType(t_int);

QName t_double = new
  QName("http://www.w3.org/2001/XMLSchema", "double");
call.addParameter("double_1", t_double,
                  ParameterMode.IN);
```

¿Preguntas...?