Servicios Web

Sesión 1: Introducción. Invocación de Servicios Web SOAP

Puntos a tratar

- ¿Qué es un Servicio Web?
- Tipos de Servicios Web
- Arquitectura de los Servicios Web
- Tecnologías básicas para Servicios Web
- Interoperabilidad de los Servicios Web
- Servicios Web desde la vista del cliente
- Invocación de servicios JAX-WS

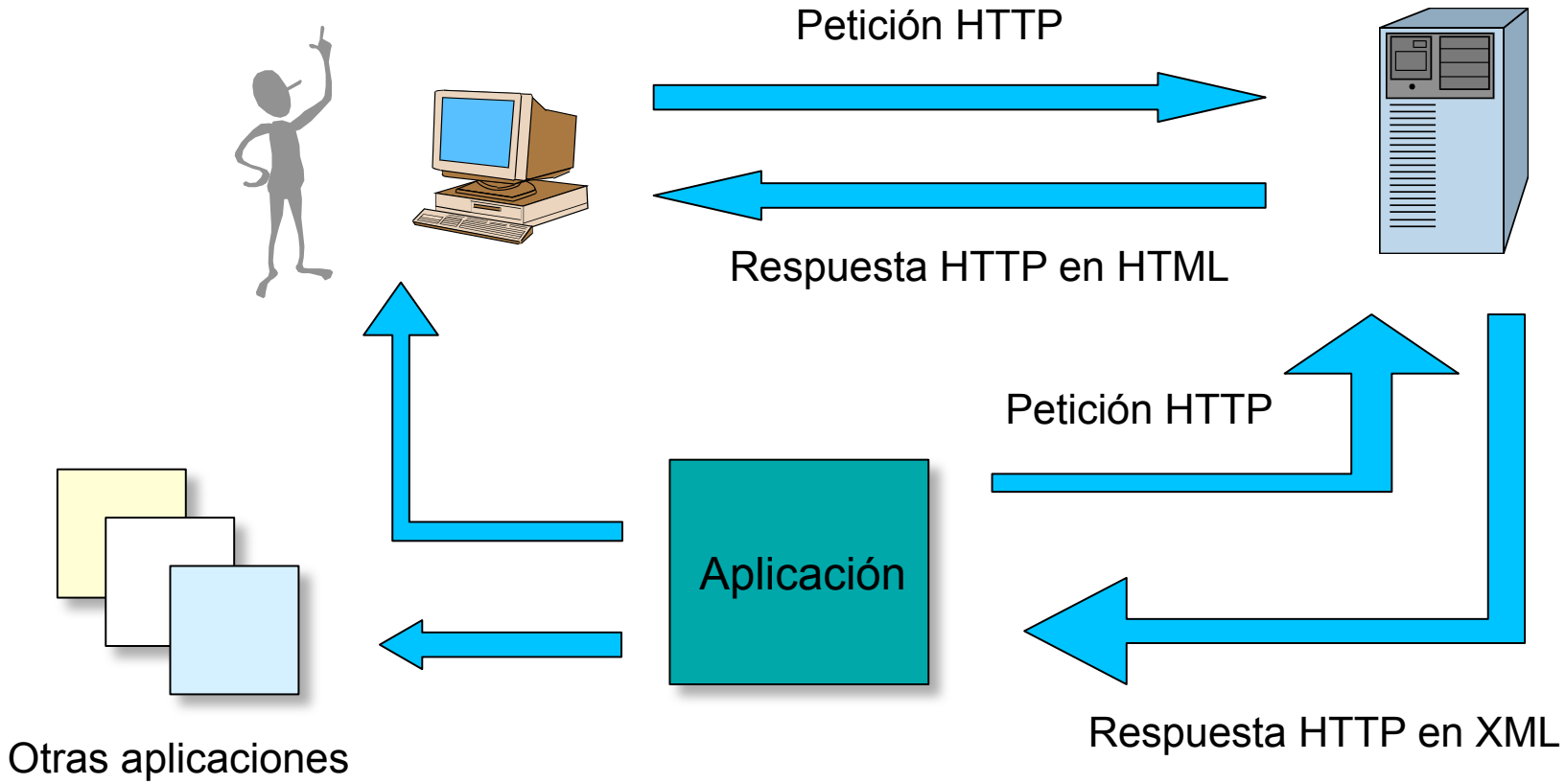
Descripción de Servicio Web (WS)

- Un Servicio Web es un componente diseñado para soportar interacciones máquina a máquina a través de la red
  - El intercambio de información se lleva a cabo mediante mensajes codificados 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
- Los servicios web son independientes de la plataforma y del lenguaje de programación en el que estén implementados
  - Nos permiten integrar aplicaciones
- Pueden combinarse con muy bajo acoplamiento para conseguir la realización de operaciones complejas proporcionando un valor de negocio añadido

La cuestión clave es la interoperabilidad!!

- Su principal característica es su gran INTEROPERABILIDAD y extensibilidad
  - Los servicios Web fueron “inventados” para solucionar el problema de la interoperabilidad entre las aplicaciones
  - El uso de XML hace posible el compartir datos entre aplicaciones con diferentes plataformas hardware y proporciona información fácilmente procesable por las máquinas (“web para máquinas”)
  - El protocolo HTTP asegura que puedan ser llamados por cualquier aplicación (independientemente del lenguaje de programación y sistema operativo)
- Las características deseables de un Servicio Web son:
  - Los servicios web deben ser accesibles a través de la red, deben contener una descripción de si mismos, y deben poder ser localizados

Web “para humanos” vs. web “para máquinas”

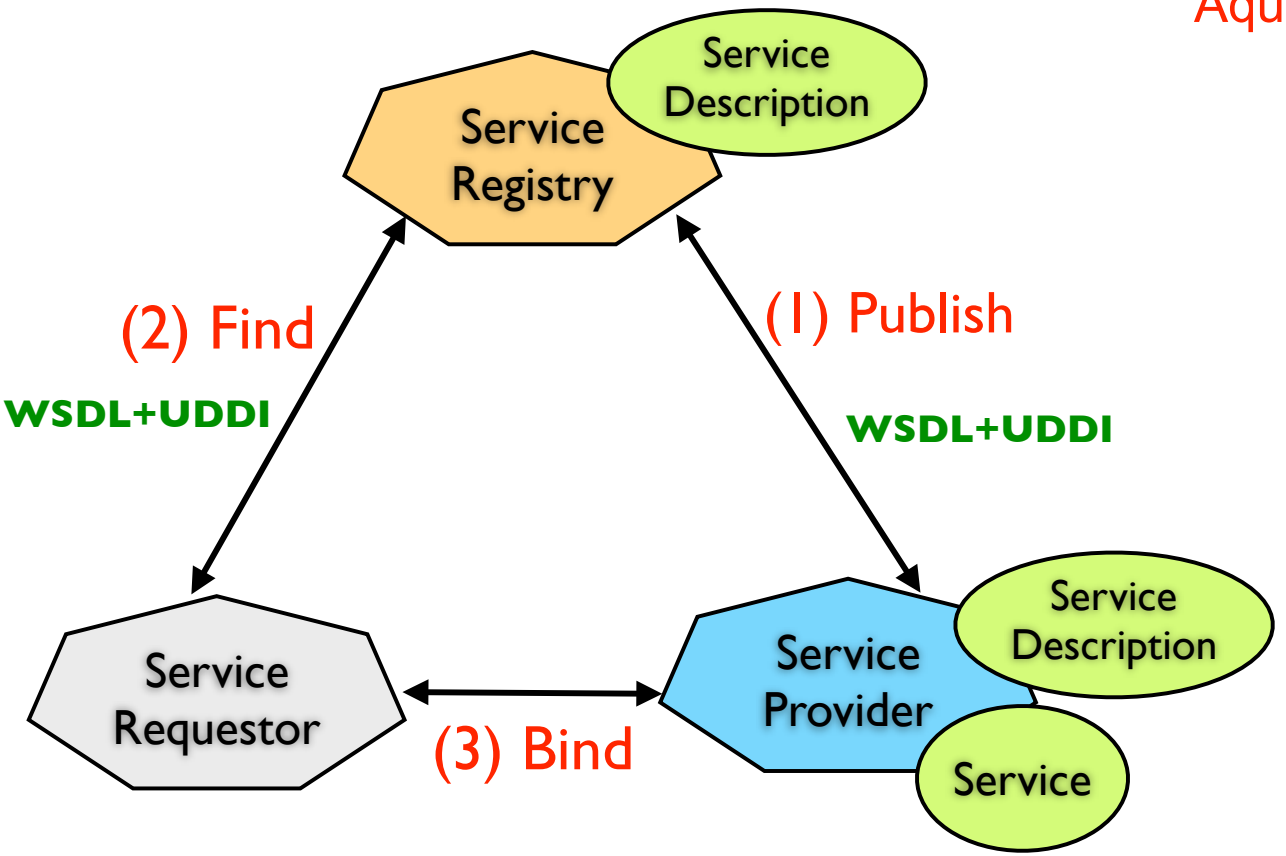


Tipos de Servicios Web

- A nivel CONCEPTUAL, un servicio es un componente software proporcionado a través de un *endpoint* accesible a través de la red. Los productores y consumidores de servicios utilizan mensajes para intercambiar información.
- A nivel TÉCNICO, los servicios pueden implementarse de varias formas:
  - Servicios Web SOAP
    - Utilizan mensajes XML que siguen el estándar SOAP
    - Describen su interfaz utilizando WSDL
  - Servicios Web RESTful
    - Utilizan estándares muy conocidos: HTTP, URI, MIME
    - Tienen una infraestructura muy “ligera”

Arquitectura de los servicios Web

- Un WS normalmente reside en una máquina remota y es llamado por un cliente a través de la red



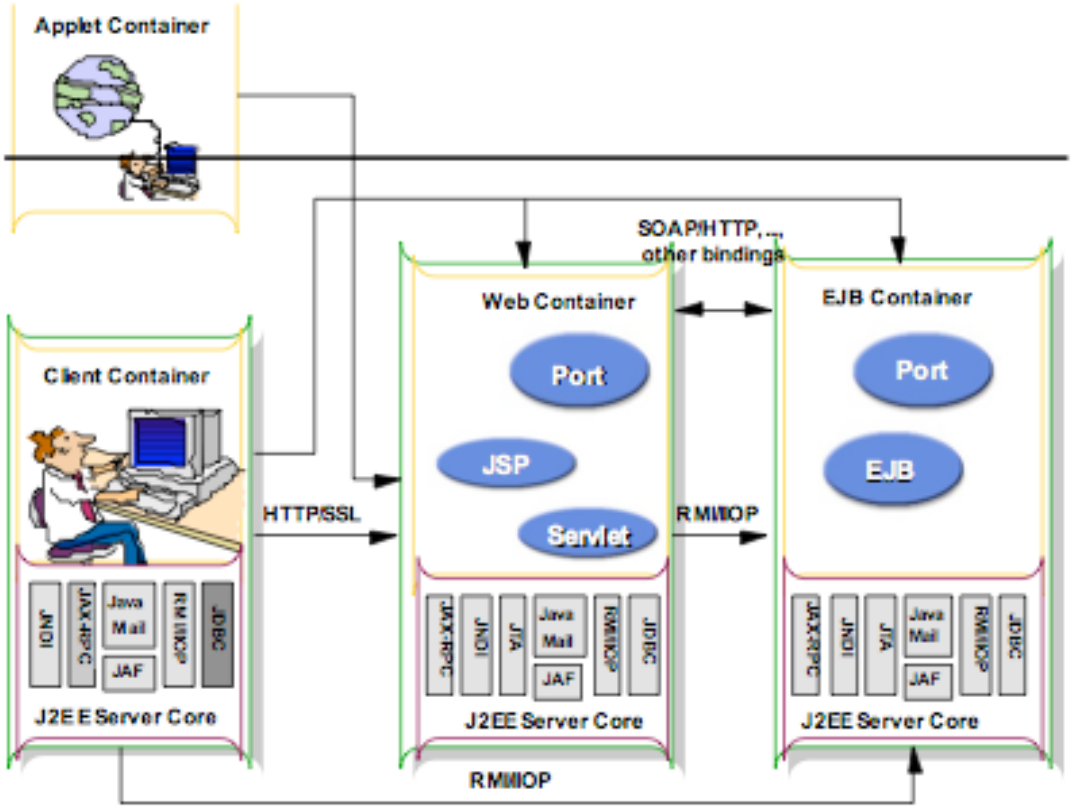
Aquitectura orientada a servicios

PERMITE:

- crear una descripción abstracta del servicio
- proporcionar una implementación concreta
- publicar y localizar un servicio
- seleccionar una instancia del servicio
- utilizar dicho servicio

Servicios web SOAP y JavaEE

- La especificación que define los servicios Web para Java EE (JSR-109) requiere que una instancia de un servicio (denominada Port, o componente Port) sea creada y gestionada por un contenedor. Este Port puede ser referenciado desde un cliente, así como desde los contenedores web y EJB



Pueden implementarse de dos formas:

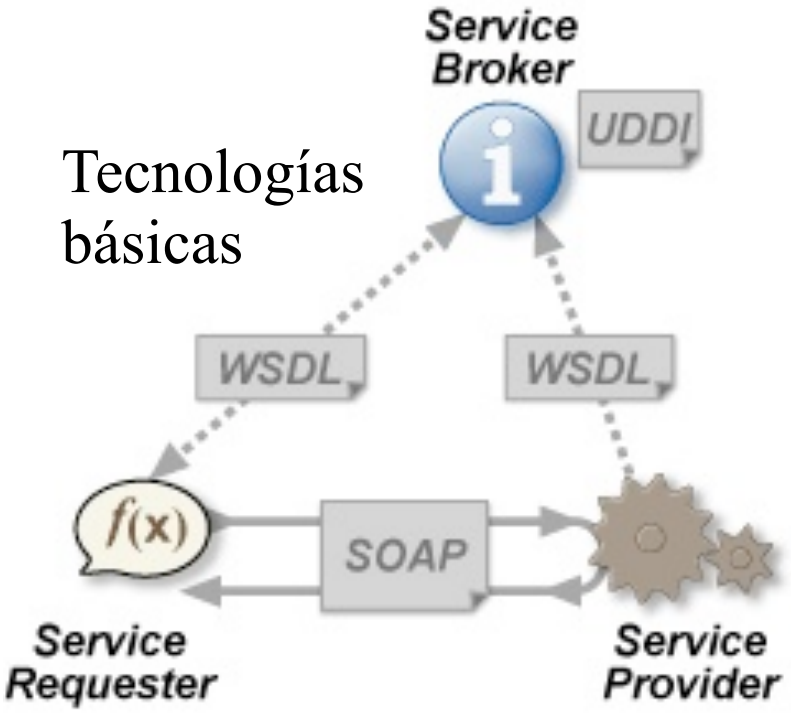
- como una clase Java que se ejecuta en un contenedor Web
- como un EJB de sesión o *singleton* en un contenedor EJB

El contenedor actúa como mediador para acceder al servicio

Tecnologías básicas para servicios Web

- Los protocolos utilizados en los WS se organizan en una serie de capas:

Tecnologías básicas



Localización de Servicios (*UDDI*)
Descripción de Servicios (*WSDL*)
Mensajería XML (*SOAP*, ***XML-RPC***)
Transporte de Servicios (*HTTP*, ***SMTP***, ***FTP***, ***BEEP***, ***...***)

SOAP

- Protocolo derivado de XML
- Se usa para intercambiar información entre aplicaciones
- 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 primer 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

Ejemplos mensajes SOAP

- Mensaje de petición

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
          soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

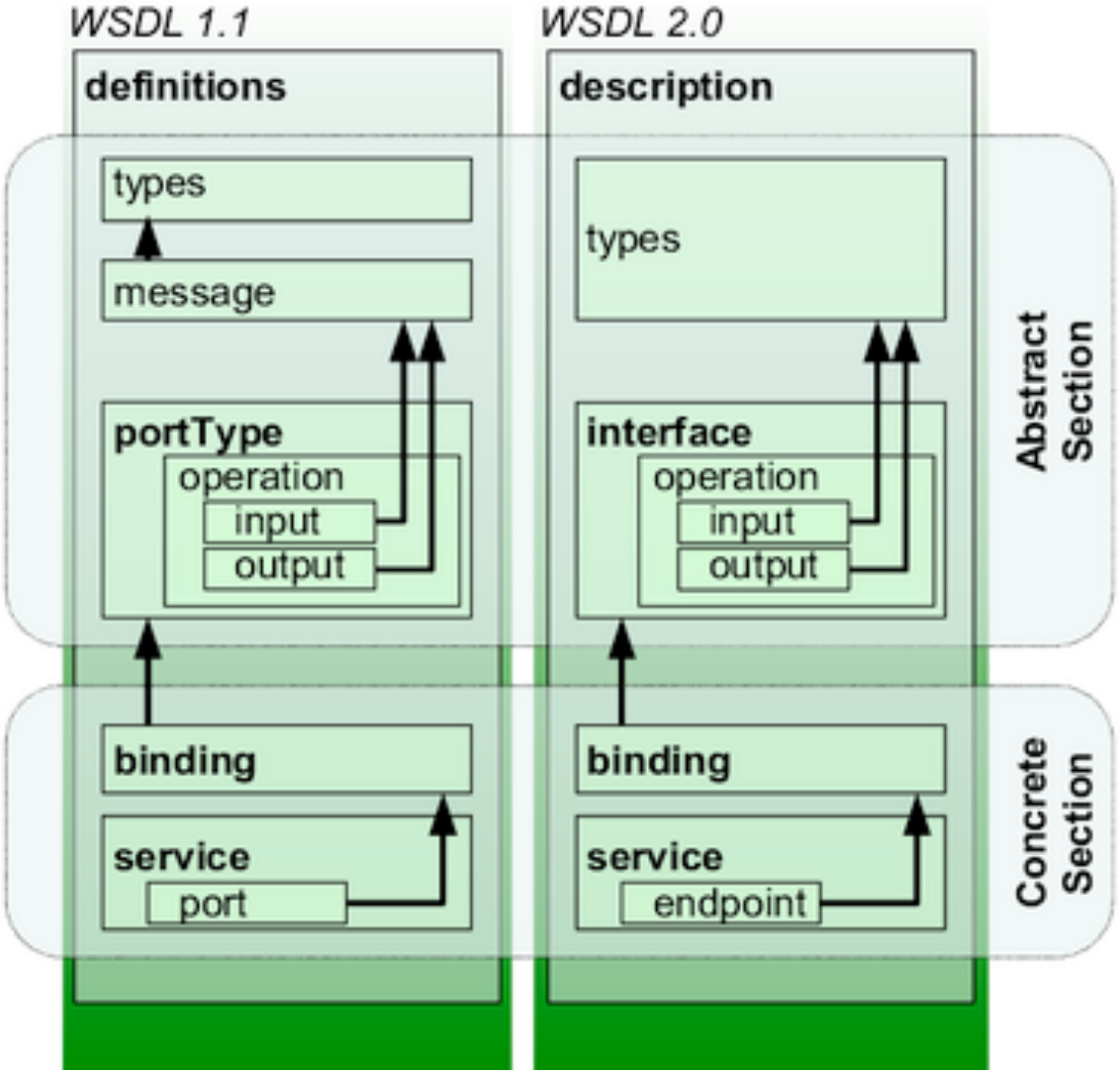
- Mensaje de respuesta

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
           soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>
</soap:Envelope>
```

WSDL (Web Services Description Language)

- 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

Estructura de un documento WSDL



La parte abstracta define el QUÉ hace el servicio:

- operaciones disponibles
- entradas, salidas y mensajes de error
- definiciones de tipos para los mensajes

La parte concreta define el CÓMO Y DÓNDE del servicio:

- cómo se tiene que llamar (formato de los datos: SOAP)
- protocolo de acceso (red)
- dónde está el servicio (URL)

Elementos WSDL (versión 1.1)

<definitions>

<types> *tipos de datos, si no son primitivos*

<message> *llamadas y respuestas SOAP*

<portType> ***(INTERFAZ)*** *operaciones: llamada + respuesta*

<binding> *protocolo de red y formato de datos SOAP*

<service> *URL del servicio para acceder a una colección de ports*

Ejemplo de documento WSDL (I)

```
<?xml version="1.0" encoding="utf-8"?>

<definitions targetNamespace="http://jaxwsHelloServer/"
name="HelloService">
  <types>
    <xsd:schema>
    <xsd:import namespace="http://jaxwsHelloServer/"
      schemaLocation="http://localhost:8080/JAXWSHelloAppServer/
                                        jaxwsHello?xsd=1"/>
    </xsd:schema>
  </types>

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

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

los tipos se definen en el fichero xsd

Ejemplo de documento WSDL (II)

```
<portType name="Hello">
    <operation name="sayHello">
       <input wsam:Action="http://jaxwsHelloServer/Hello/sayHelloRequest"
           message="tns:sayHello"/>
       <output wsam:Action="http://jaxwsHelloServer/Hello/sayHelloResponse"
           message="tns:sayHelloResponse"/>
    </operation>
</portType>

<binding name="HelloPortBinding" type="tns:Hello">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                 style="document"/>
     <operation name="sayHello">
      <soap:operation soapAction=""/>
        <input> <soap:body use="literal"/> </input>
        <output> <soap:body use="literal"/> </output>
     </operation>
</binding>

<service name="HelloService">
  <port name="HelloPort" binding="tns:HelloPortBinding">
     <soap:address location="http://localhost:8080/JAXWSHelloAppServer/
                                                   jaxwsHello"/>
  </port>
</service>
```

operaciones soportadas por el servicio

protocolo de red y formato de los datos

dirección donde localizar el servicio

Edición de documentos WSDL con Netbeans

- Para poder editar documentos WSDL hay que instalar el plugin XML desde:
  - `URL: http://deadlock.netbeans.org/hudson/job/xml/lastSuccessfulBuild/artifact/build/updates/updates.xml`
- Este plugin nos permitirá trabajar con ficheros WSDL y con ficheros de esquema (xsd)
- ***XML Schema*** es una recomendación del W3C, que proporciona mecanismos para definir la estructura, contenido y semántica de un documento xml
  - Un documento WSDL utiliza ficheros de esquema para definir los TIPOS de mensajes que se utilizan como interfaz para comunicarnos con un servicio Web

Fichero de esquema (xsd)

- El bloque de construcción principal de un documento xml es ***<element>***, que debe contener
  - Una propiedad ***name*** que representa el nombre del elemento
  - Una propiedad ***type*** para indicar el tipo de elemento
  - Podemos utilizar alguno de los tipos predefinidos (*built-in types*), o bien podemos definir nuevos tipos utilizando etiquetas ***simpleType*** o ***complexType***
- Ejemplos:

```
<xs:element name="CustomerAddress" type="xs:string"/>
```

Ejemplos de definiciones de esquema (xsd)

- Definimos el elemento CustomerAddress

```
<xs:element name="CustomerAddress" type="xs:string"/>
```

- Podemos utilizar dicho elemento en el documento wsdl

```
<message name="msgResponse">
     <part name="parameters" element="tns:CustomerAddress"/>
</message>
```

- Ejemplo de mensaje de respuesta con la definición anterior:

```
<Customer_address>Calle de los Pinos, 37</Customer_address>
```

Ejemplos de definiciones de esquema (xsd)

- Vamos a definir nuevos tipos:

```
<xsd:element name="Customer" type="tns:CustomerType"/>
<xsd:complexType name="CustomerType">
  <xsd:sequence>
    <xsd:element name="Phone" type="xsd:integer"/>
    <xsd:element name="Addresses" type="tns:AddressType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AddressType">
  <xsd:sequence>
    <xsd:element name="Address1" type="xsd:string"/>
    <xsd:element name="Address2" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

- Definición del mensaje en el WSDL:

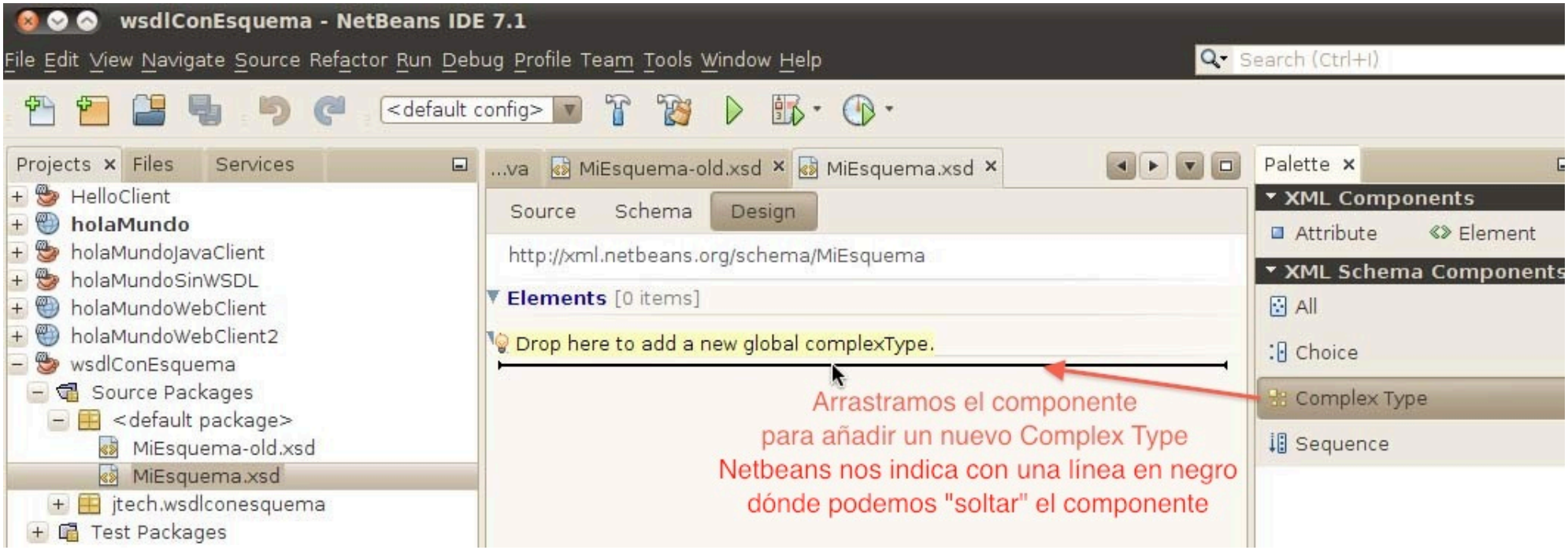
```
<message name="msgResponse2">
     <part name="parameters" element="tns:Customer"/>
</message>
```

- Ejemplo de mensaje de respuesta:

```
<Customer>
   <Phone>12345678</Phone>
   <Address1>Calle de los Pinos, 37</Address1>
   <Address2>Calle de los Manzanos, 25</Address2>
</Customer>
```

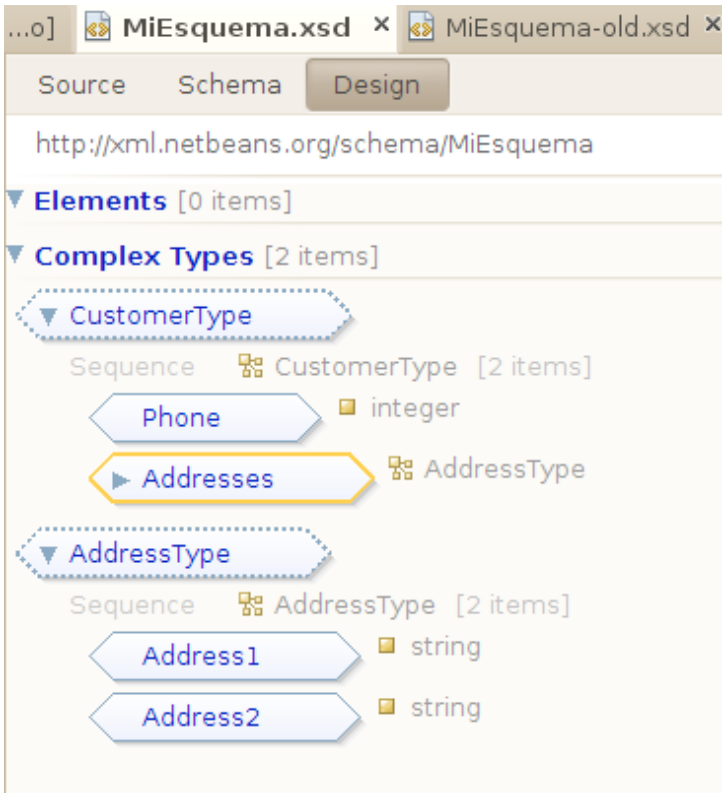
Edición de esquemas con Netbeans (I)

New->XMl ->XML Schema

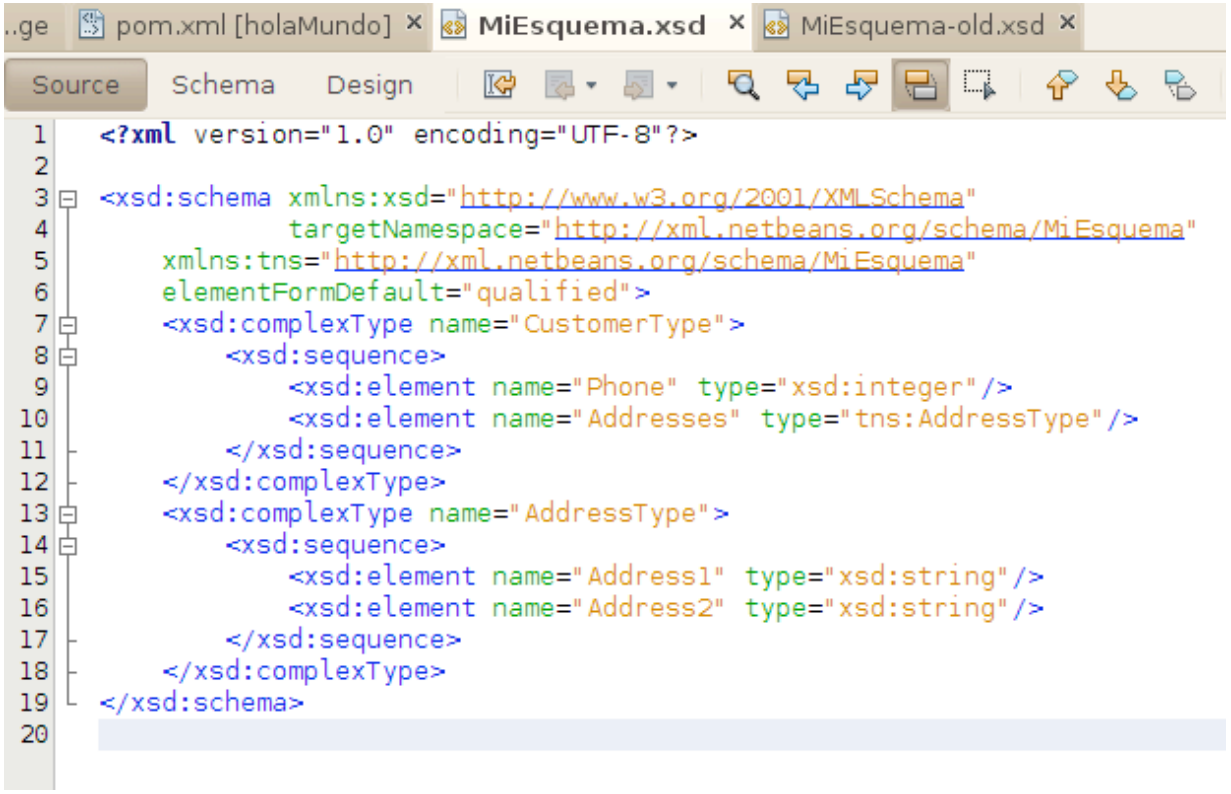


Edición de esquemas con Netbeans (II)

Vista de diseño

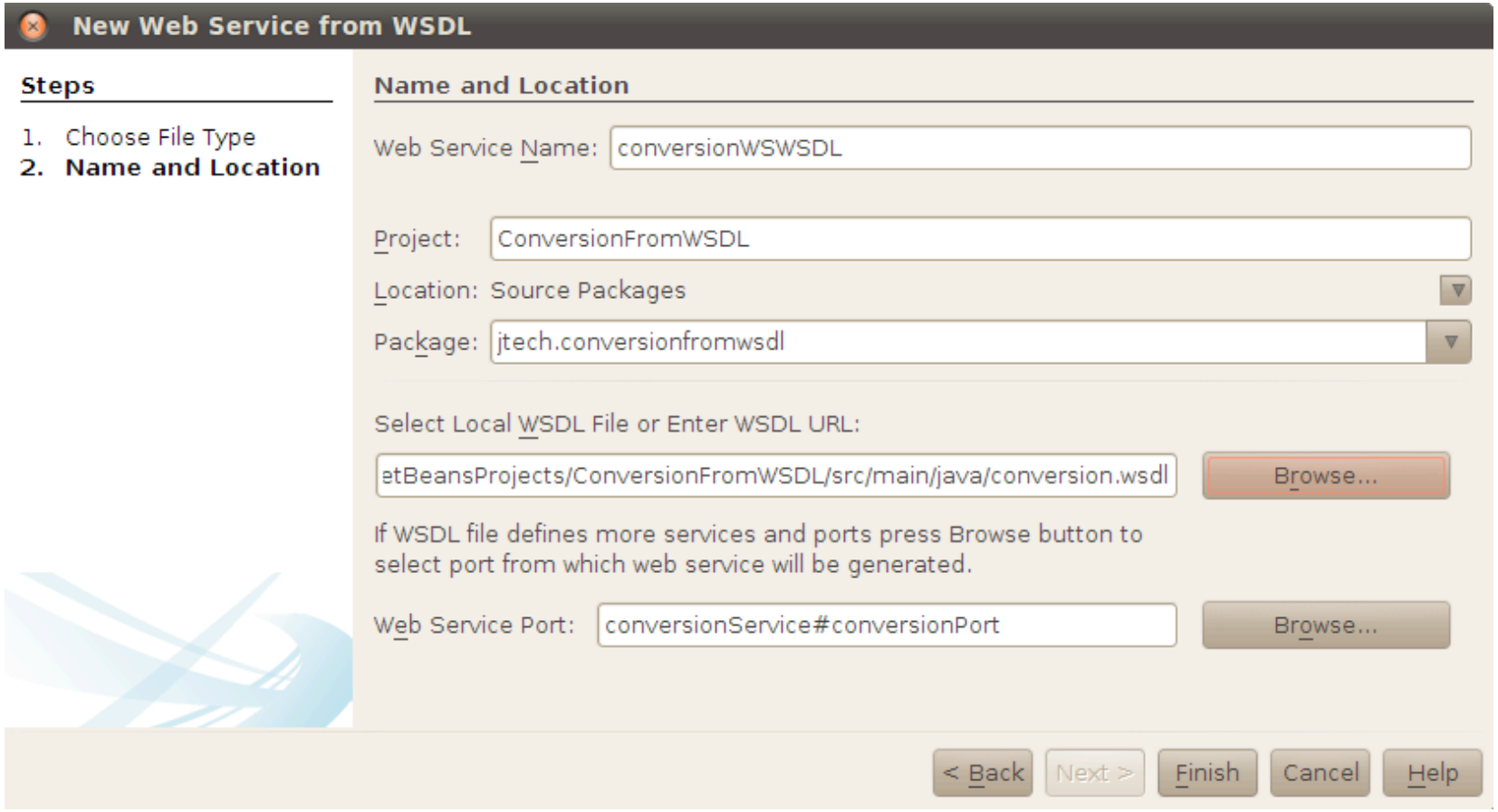


Vista de fuentes



Edición de WSDL con Netbeans (I)

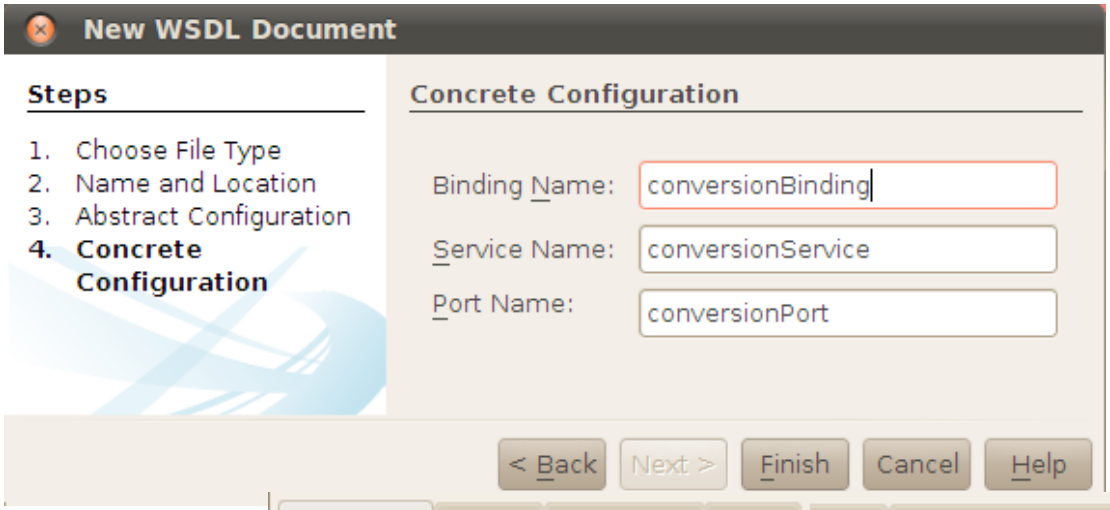
New->XMl ->WSDL Document



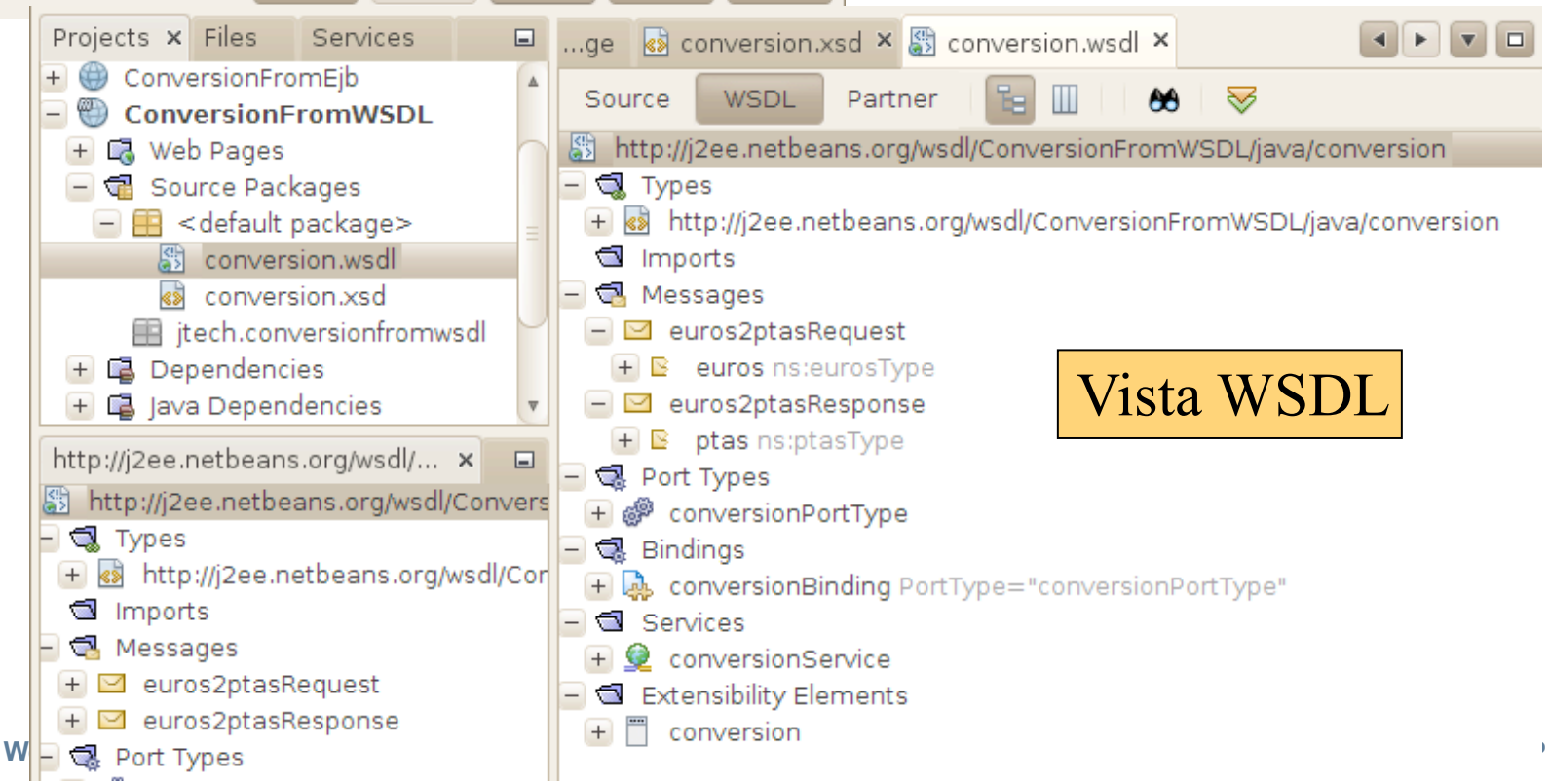
Edición de WSDL con Netbeans (II)

Configuración ABSTRACTA

Edición de WSDL con Netbeans (III)



Configuración CONCRETA



Vista WSDL

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

Interoperabilidad de los servicios Web (WS-I): Metro y JAX-WS

METRO: Iniciativa de Sun para conseguir la interoperabilidad de los Servicios Web

- Metro/WSIT
- JAXWS-RI
- JAXB-RI

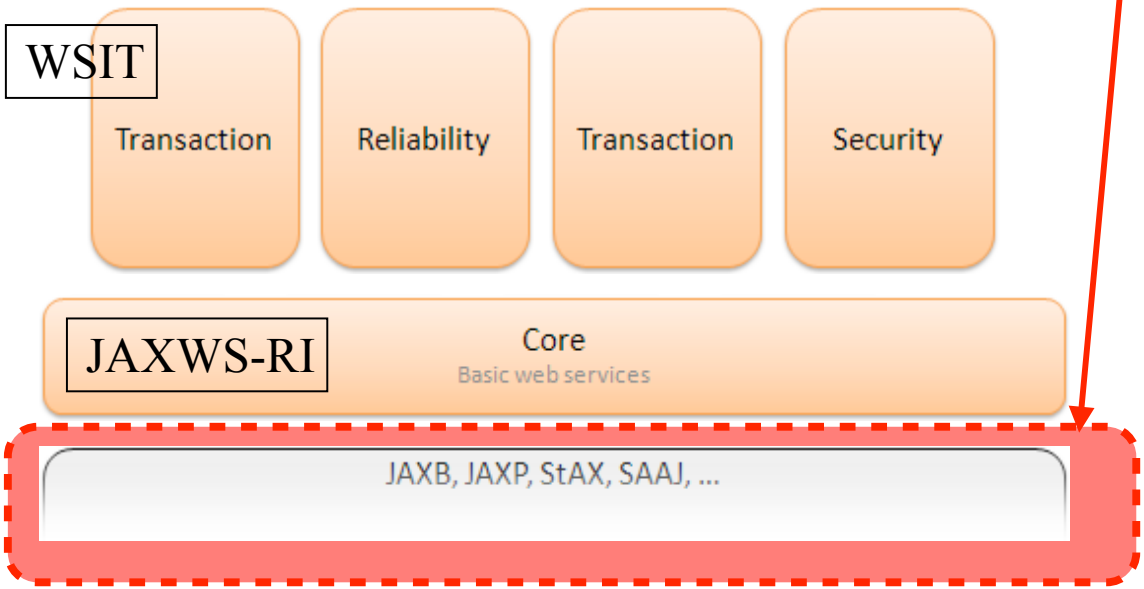
WSIT: Web Services Interoperabe Technologies (permiten la interoperabilidad con .NET)
Transport: HTTP, MTOM, SOAP/TCP
Reliability: WS-ReliableMessaging; WS-Coordination; WS-Atomic Transaction
Security: WS-Security; WS-Trust
Bootstrapping: WSDL; WS-Policy; WS-MetadataExange

Implementación de Referencia del API JAX-WS (JSR-224: Java Api for XML-based Web Services)
Estándares asociados: WS-I Basic Profile (SOAP y UDDI); WS-I Attachment Profile (SOAP con anexos); WS-I Addressing (espacios de nombres y ficheros de esquema)

Implementación de Referencia del API JAX-WS (JSR-222: Java Architecture for XML Binding (JAXB) 2.0)

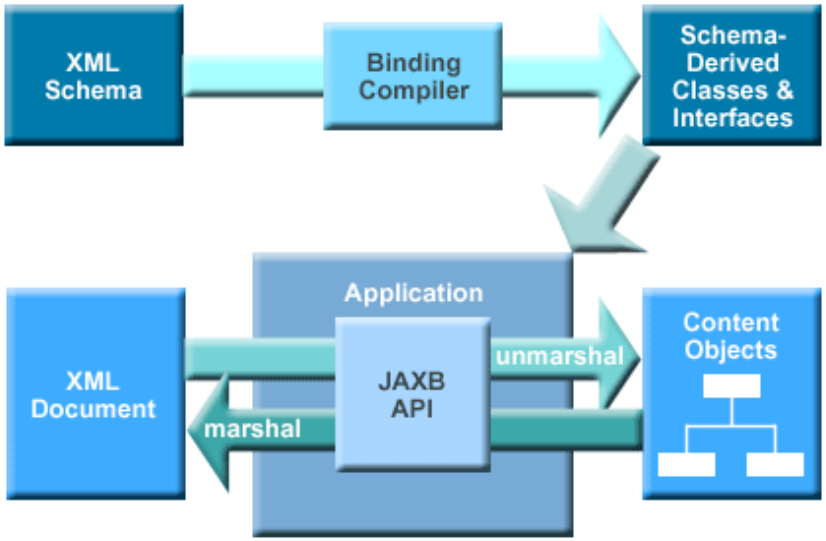
Interoperabilidad de los servicios Web (WS-I): Metro y JAX-WS

METRO: Iniciativa de Sun para conseguir la interoperabilidad de los Servicios Web



JAXB-RI

Implementación de referencia del API **JAXB** (Java Architecture for XML Binding)



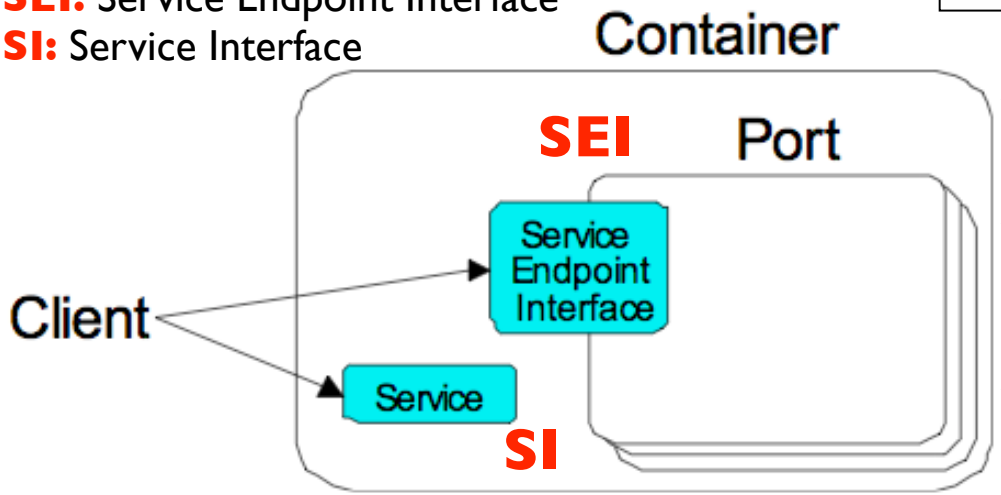
Metro está construido sobre un conjunto de librerías que pueden usarse de forma independiente fuera del contexto de los Servicios Web

Servicios Web: vista del cliente

El transporte, codificación y dirección del Port son **transparentes** para el cliente

JAX-WS proporciona una factoría (Service) para seleccionar qué Port desea usar el cliente.
La clase **Service** proporciona los métodos para acceder al Port

SEI: Service Endpoint Interface
SI: Service Interface




```
...
//Primero accedemos al objeto Service
Hola_Service service = new Hola_Service();

//a través de él accedemos al Port
Hola port = service.getHolaPort();
java.lang.String name = "perico de los palotes";

//utilizamos el Port para llamar al WS a través
del SEI
java.lang.String result = port.hello(name);
System.out.println("Result = "+result);
...
```

Tipos de clientes:

- aplicación cliente Java EE
- componente Web
- componente EJB
- otro Servicio Web

El **cliente**, para acceder al servicio, necesita:
(1) Acceder a la clase/interfaz Service (**SI**)
(2) A través del SI obtiene el **Port**
(3) Realizar llamadas a métodos del **SEI** del Port correspondiente

Tipos de acceso a servicios Web

- **JAX-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-WS

- La versión actual de JAX-WS es la 2.2, también denominada JSR-224
- La implementación de referencia de JAX-WS está enmarcada dentro del proyecto Metro
- A partir de JDK 1.6 se incluye JAX-WS 2.0 en Java SE
  - JAX-WS 2.1 a partir de JDK 1.6.0_04
- JAX-WS también viene incluida en Glassfish
  - JAX-WS 2.2 en Glassfish 3.1.2.2
  -

Generar el cliente con JAX-WS y JDK 1.6

- 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}" />
```

- Y también desde Maven ...!

Cliente de un SW desde una clase Java con Maven

- Necesitamos conocer la dirección del WSDL (o disponer de él en local)
  - la etiqueta *<portType>* nos indica las operaciones que el servicio ofrece, con los mensajes de entrada y salida. Representa el SEI del servicio

```
<types>
   <xsd:schema>
   <xsd:import namespace="http://sw/"
     schemaLocation="http://localhost:8080/HolaMundo/hola?xsd=1"/>
   </xsd:schema>
 </types>
...
  <portType name="Hola">
    <operation name="hello">
       <input wsam:Action="http://sw/hola/helloRequest"
           message="tns:hello"/>
       <output wsam:Action="http://sw/Hola/helloResponse"
           message="tns:helloResponse"/>
    </operation>
 </portType>
...
```

definiciones de los tipos utilizados en los mensajes

nombre de la operación

mensaje de entrada

mensaje de salida

Los mensajes

- Los mensajes pueden tener parámetros, cuyos tipos se especifican en el fichero de esquema declarado en la etiqueta <*types*> del wsdl

tipo del parámetro

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

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

tipo del parámetro

El fichero de esquema

- Define los tipos de los parámetros utilizados en los mensajes
  -

```
<xs:schema version="1.0" targetNamespace="http://sw/">
   <xs:element name="hello" type="tns:hello"/>
   <xs:element name="helloResponse" type="tns:helloResponse"/>

   <xs:complexType name="hello">
     <xs:sequence>
       <xs:element name="arg0" type="xs:string" minOccurs="0"/>
     </xs:sequence>
   </xs:complexType>

   <xs:complexType name="helloResponse">
      <xs:sequence>
         <xs:element name="return" type="xs:string" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>...
```

cadena de caracteres

cadena de caracteres

Pasos a seguir

- Creamos un proyecto Java simple con Maven
- Editamos el *pom.xml* para añadir las dependencias necesarias
- Codificamos la clase cliente, utilizando las clases generadas por *wsimport*
- Empaquetamos y ejecutamos el cliente

Edición del pom.xml

- Incluimos la dependencia con la librería *webservices-rt*:

```
<dependencies>
  ...
  <dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>webservices-rt</artifactId>
    <version>1.4</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

- Incluimos el plugin *jaxws* para ejecutar *wsimport*
  - **jaxws: wsimport** lee un fichero WSDL y genera las clases necesarias para la creación, despliegue e invocación del servicio web

Plugin *jaxws*

```
<plugins>
    <plugin>
       <groupId>org.jvnet.jax-ws-commons</groupId>
       <artifactId>jaxws-maven-plugin</artifactId>
       <version>2.2</version>
       <executions>
          <execution>
              <goals> <goal>wsimport</goal> </goals>
          </execution>
       </executions>
       <configuration>
         <packageName>wsClient</packageName> <!--opcional-->
         <wsdlUrls>
           <wsdlUrl>http://localhost:8080/HolaMundo/hola?wsdl</wsdlUrl>
         </wsdlUrls>
         <verbose>true</verbose>
       </configuration>
       <dependencies>
         <dependency>
             <groupId>javax.xml</groupId>
             <artifactId>webservices-api</artifactId>
             <version>1.4</version>
         </dependency>
       </dependencies>
     </plugin>
     ...
```

meta wsimport

paquete en el que se generarán las clases

ubicación del wsdl

Clases generadas por *wsimport*

- Las clases se generarán cuando compilemos nuestro proyecto (en *target/generated-sources/)*
  - Hola.java: interfaz del servicio (etiqueta wsdl <portType>)
  - Hola_Service.java: clase que representa el servicio, contiene el método *getHelloPort* para acceder a la operación del servicio. También contiene la url del servicio
  - ObjectFactory.java: factoría de métodos para obtener representaciones java a partir de definiciones XML
  - Hello.java, HelloResponse.java: clases que representan los tipos definidos en el wsdl (utilizan anotaciones JAXB)
  - package-info.java

Código para acceder al servicio

- El cliente NO tiene control sobre el ciclo de vida del servicio

```
package expertoJava;

public class App
{
    public static void main( String[] args )
    {
        sw.Hola_Service service = new sw.Hola_Service();
        sw.Hola port = service.getHolaPort();
        System.out.println(port.hello(" amigos de los
                                    Servicios Web!"));

    }
}
```

clase utilizada para recuperar el proxy del servicio

interfaz del servicio: SEI

proxy: representante local del servicio remoto

invocamos el método que ofrece el servicio

Código de acceso desde un servlet/jsp

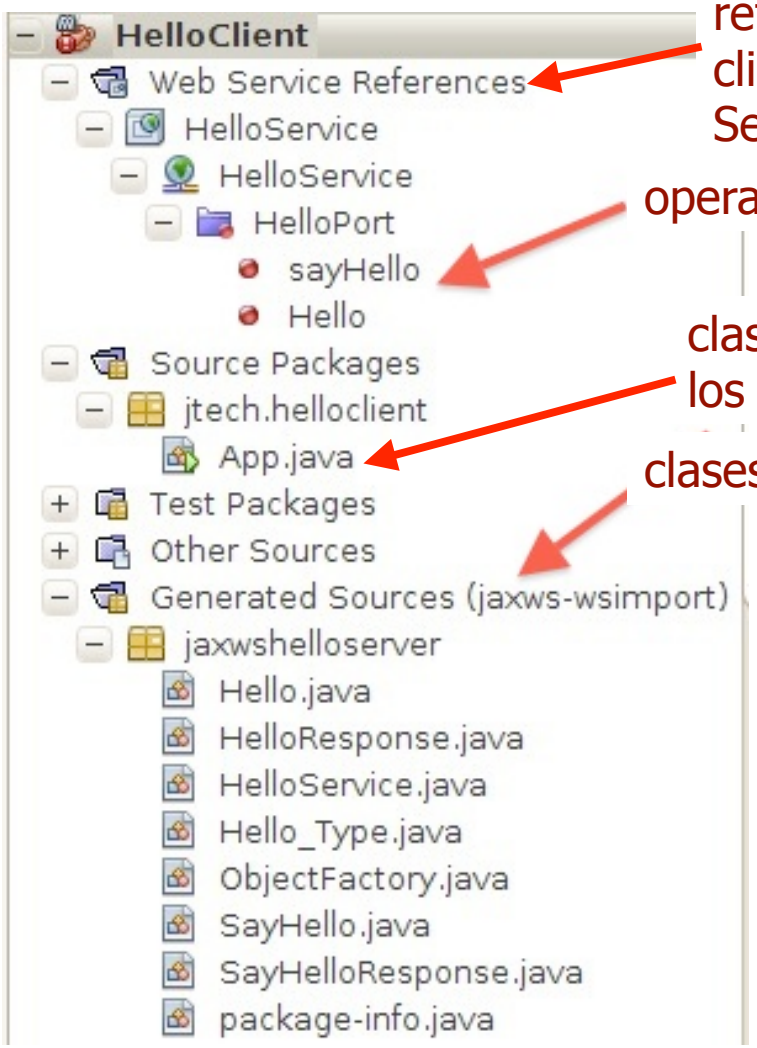
```
@WebServlet(name = "Hola", urlPatterns = {"/Hola"})
public class NewServlet extends HttpServlet {

    @WebServiceRef
    private Hola_Service service;

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
                                throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            ...

            try { // Obtenemos el Port
                sw.Hola port = service.getHolaPort();
                java.lang.String name = "amigos de los Servicios Web";
                // Llamamos a la operación correspondiente del SEI
                java.lang.String result = port.hello(name);
                out.println("Result = "+result);
            } catch (Exception ex) {
              // Manejo de excepciones
            }
           ...
```

Java EE6 puede inyectar una referencia a un servicio con la anotación @WebServiceRef

Invocación de Servicios Web con Netbeans

- Desde Netbeans podemos crear un cliente de un servicio Web de forma sencilla
- Pasos a seguir:
  - Desde un proyecto nuevo (o ya existente), añadimos un *stub* con: *New > Web Service Client*
    - Se abrirá un asistente en el que indicaremos la URL del servicio Web, la librería para realizar las llamadas, ...
  - Una vez creado el *stub*, añadiremos el código para acceder al servicio mediante: *Insert Code... > Call Web Service Operaion*
    - Podemos hacerlo desde la clase principal, o desde otra clase, incluso desde un JSP

Vista de un cliente de un WS con Maven



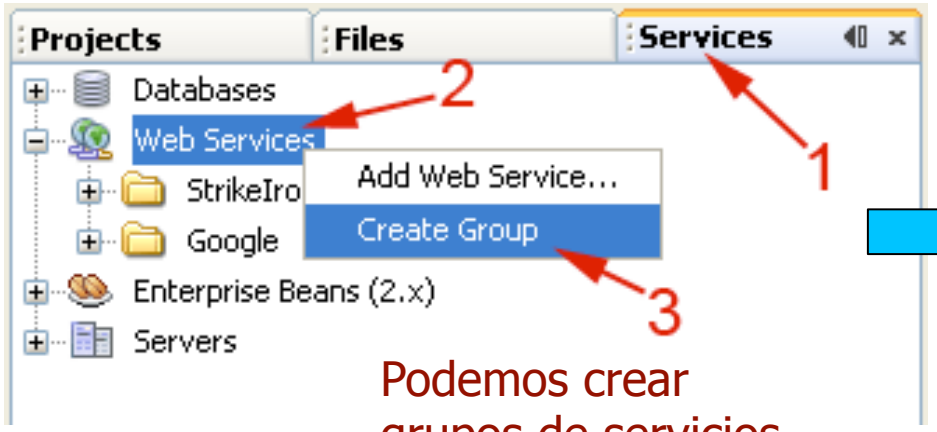
referencias de servicios Web contenidas en el cliente del servicio (creadas con New->Web Service client)

operaciones que ofrece el servicio

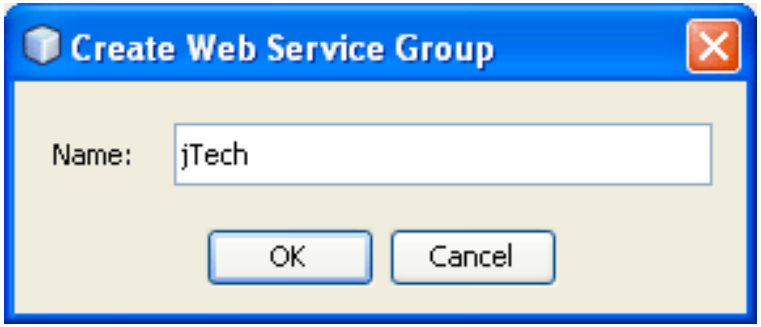
clase que contiene la invocación de los métodos del servicio

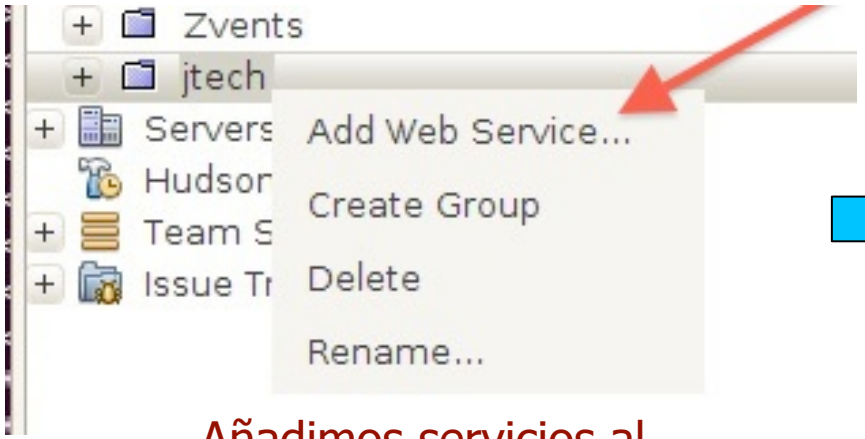
clases generadas por wsimport

Gestor de servicios de Netbeans

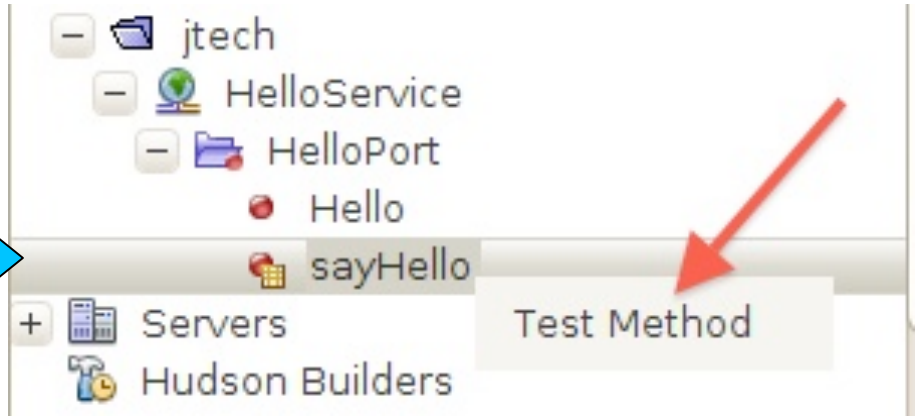


Podemos crear grupos de servicios





Añadimos servicios al grupo



Podemos probar servicios desde el gestor de servicios de Netbeans

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-WS
  - Perdemos totalmente la transparencia
- JAX-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...?