



WSDL (Web Services Description Language) Construcción de servicios web

Lourdes Tajés Martínez
(tajes@uniovi.es)



En 5 minutos...

- Un servicio web se define (en palabras del [W3C](#)) como una aplicación software, identificada por un URI que se puede definir, describir y descubrir a través de documentos XML. Soporta la interacción directa con otros agentes software usando mensajes basados en documentos XML intercambiados vía protocolos de Internet.
 - Usualmente, el término se refiere a clientes y servicios que se comunican usando mensajes SOAP sobre HTTP.
 - Se asume que hay también una ***machine readable description*** de las operaciones soportadas por el servicio.
 - No es un requerimiento obligatorio **PERO** permite la generación automática de código entre cliente y servicio



Web Service Specifications (WS-*)

- Descripción de servicio: WSDL, ...
- Mensajería y llamada a funciones
 - SOAP, SOAP con attachments, SOAP sobre UDP, XML-RPC, WS-Addressing, ...
- WS-I: Mejorar la interoperabilidad entre implementaciones de distintos vendedores
- Security: XML-Signature, XML Encryption, WS-Security, WS-Trust
- Transaction: WS-Coordination, WS-Transaction, ...



Web Service Protocols

- SOAP (XML-RPC)
- REST (REpresentational State Transfer)
- **WSDL**
- UDDI (Universal Description, Discovery and Integration)
- BPEL (Business Process Execution Language)



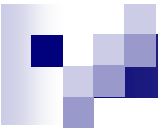
Web Service Frameworks

Nombre	Plataforma	Destino	Especificaciones	Protocolos
Apache Axis	Java/C++	Cliente Servicio	WS-ReliableMessaging WS-Coordination WS-Security WS-AtomicTransaction WS-Addressing	SOAP WSDL
Java Web Service Development Pack	Java	Cliente Servicio	WS-Security WS-Addressing	SOAP WSDL
XFire	Java	Cliente Servicio	WS-Security WS-Addressing	SOAP WSDL
gSOAP	C/C++	Cliente Servicio	WS-Discovery WS-Security WS-Enumeration WS-Addressing	SOAP XML-RPC WSDL

¿Por qué hay que definir los servicios web?


- Los servicios web exponen un sistema software con el que las **aplicaciones cliente** pueden interaccionar a través de la red.
- Una visión orientada al software
 - Métodos a invocar, con qué parámetros, cuáles son los valores de retorno, las excepciones que se propagan si algo va mal, ...
 - Casi lo mismo con lo que llevamos tiempo trabajando. Pero
 - muy lejos (internet mediante)
 - No conocemos el detalle del servicio. No tenemos acceso a su código fuente. No ha sido hecho para nosotros.





¿Por qué hay que definir los servicios web?

- **Pero**, para que tal interacción se lleve a cabo de forma satisfactoria, el servicio debe ser descrito y publicitado a sus consumidores potenciales.
- Los consumidores deben ser capaces de encontrar una descripción de cómo interaccionar con el servicio:
 - ¿ cómo se llaman los métodos? ¿qué datos espera recibir (parámetros)? ¿devuelve algún resultado (tipo de retorno)?
 - ¿qué protocolos de transporte soporta (http, smtp, ppp, ...)?
 - ¿formato de los mensajes?




¿Por qué hay que definir los servicios web? (5 minutos)

■ Alumnos DNI PAR

- ☐ Hotel Rural
- ☐ Función de comprobación de disponibilidad para unas fechas y un número de personas determinadas

■ Alumnos DNI IMPAR

- ☐ Central de reservar rurales
- ☐ Consulta de la disponibilidad de todos los hoteles para dar al cliente el mejor precio



¿Por qué hay que definir los servicios web? (5 minutos)

■ Alumnos DNI PAR

- ☐ ¿Qué información deberías darle a la central de reservas?

■ Alumnos DNI IMPAR


- ☐ ¿Qué requieres de los hoteles?

Age Group	Percentage
18-24	35%
25-34	25%
35-44	15%
45-54	10%
55-64	8%
65-74	5%
75-84	3%
85+	2%

- ☐ El servicio puede atender peticiones de consulta de disponibilidad
 - ☐ Debe decir qué número de personas, de cuándo a cuándo van a alojarse.
- Hay que ponerse de acuerdo en...
- ☐ Nombre del método
 - ¿ Se llama Disponibilidad? ¿ConsultaDisponibilidad? ¿Availability?
 - ☐ Y en los Parámetros
 - El periodo de alojamiento ¿ cómo se indica? ¿día entrada, día salida?, ¿día entrada, número de días?
 - ☐ Y el formato del mensaje
 - ¿Cómo sabe el cliente qué tipo de mensaje enviar?
 - SOAP es sólo el formato, pero no indica qué poner en la cabecera o en el contenido
 - Y la codificación de los datos enviados ¿ Qué XML hay que poner en el cuerpo del mensaje SOAP?
 - ☐ ¿ cómo son los parámetros? ¿ Un array de Strings? ¿ Varios Strings? ☐ No lo sé ☐

¿Por qué hay que definir los servicios web?

- Hotel Rural (Casa Otilio) (*continuación*)
- Y en la Respuesta
 - ¿ Qué formato tiene la respuesta? ¿Hay respuesta?
 - ¿ Nos devuelve sí/no? ¿nos devuelve el precio final que costaría la reserva? ¿Nos devuelve una estructura compleja con habitaciones, precio,...? Si nos responde que no, ¿nos devuelve no, simplemente, o alternativas de alojamiento?
- Protocolo de transporte
 - ¿ Puedo enviar un mail para solicitar el servicio? Ojo, eso implica que tengan un servidor de correo escuchando¡¡
 - HTTP, siempre está disponible
- ¿ A qué dirección? (Pero, ¿es que no sé ni eso?)



¿Por qué hay que definir los servicios web?

- Los hoteles, que describan todo lo anterior como les apetezca
- Las centrales de reserva, que describan cómo les gustaría que fuese.
- Necesito dos Hoteles Rurales voluntarios
- Necesito una central de reservas voluntaria
- ¿Coincide todo el mundo?



Todavía puede empeorar

- Hotel Rural (Casa Pepe Gotera)

- ☐ Otra casa rural con diferentes nombres de métodos, parámetros, etc...
- ☐ Cada uno lo hace como quiere

- Pero yo tengo un servicio de reservas naturales (Centro de reservas on-line “El Percebe”)

- ☐ Tengo que consultar varias casas rurales para hacer un resumen al cliente o para buscarle la mejor opción o la que mejor se le ajuste o Y quiero automatizarlo!!!
- ☐ Construir mensajes SOAP para cada servicio
- ☐ NECESITO SABER CÓMO SON CADA UNO DE ELLOS.



¿ Cómo mejorar nuestra situación?

- Tenemos varios servicios web que exponen un negocio con el que los clientes interaccionan a través de la red.
- Necesitamos un **contrato** entre el desarrollador del servicio y el desarrollador del cliente. **Fundamental**
- **Publicar**: Un proveedor comunica a los clientes cómo invocar el servicio web.
- **Encontrar**: El solicitante puede encontrar un servicio adecuado
- **Enlazar**: La descripción del servicio indica exactamente qué formato de mensaje tiene que ser enviado a qué dirección con el fin de invocar al servicio

WSDL (güisdel)

- WSDL (Web Services Description Language) es una gramática XML que se utiliza para **describir la interfaz pública** de los servicios Web. **De forma estandarizada**
 - Las descripciones generadas con tal gramática son documentos XML que describen la interfaz e implementación de los servicios web
- Un documento WSDL describe tres propiedades fundamentales de un servicio web
 - Las operaciones soportadas y qué mensajes las activan
 - El formato de los mensajes
 - Los tipos de datos especiales que se envíen se incluyen en el archivo WSDL en forma de XML Schema (ejemplo, DNI).
 - El protocolo de comunicación en el que se envía el mensaje
 - La forma en qué cada operación se compone de mensajes formateados de una forma específica y transmitidos por un protocolo concreto de red
- Mira el ejemplo del wsdl de amazon en http://www.amazon.com/E-Commerce-Service-AWS-home-page/b/ref=sc_fe_c_0_15763381_1?ie=UTF8&node=12738641&no=15763381&me=A36L942TSJ2AJA



Un servicio web bien definido

- Definición de la interfaz del servicio (**descripción abstracta**)
 - En términos de mensajes intercambiados (los necesarios para activar los métodos expone el servicio web)
 - Cómo formatear el mensaje (SOAP, HTTP, ...)
- Definición de la implementación del servicio (**definición concreta**)
 - Información de enlazado (binding)
 - Qué **protocolo** de comunicaciones utilizar
 - Cómo **implementar interacciones** sobre ese protocolo
 - Cómo **alcanzar** al servicio (dirección de red)



Terminología

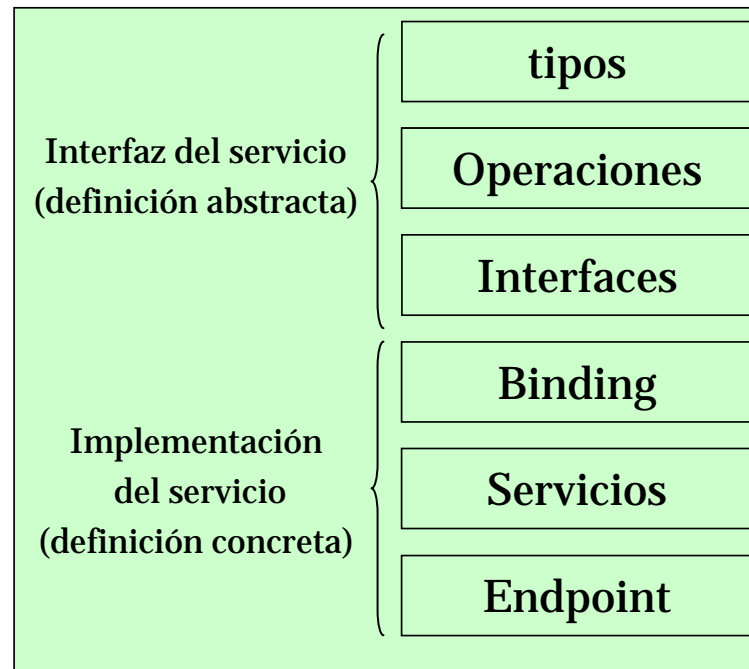
■ WSDL

- W3C Recommendation 26 June 2007
- <http://www.w3.org/TR/wsdl20/>

■ Endpoint

- Indica una localización específica para acceder a un servicio web usando un protocolo y formato de datos específico.
- Un Web service endpoint es una entidad o recurso referenciables a los que se pueden enviar mensajes. Una referencia a un endpoint debe proporcionar toda la información necesaria para direccionar un web service endpoint.

Versión 2.0 (2007)



- A caballo entre dos especificaciones: WSDL 1.X (2002) y WSDL 2.0 (2006) ya aceptada pero no muy implantada todavía (<http://www.w3.org/TR/2003/WD-wsdl12-20030303/>, <http://www.w3.org/TR/wsdl20-primer/>)

Especificación WSDL 1.2: Parte Abstracta

- La parte abstracta indica {interfaz, mensajes, tipos de parámetros}
 - Parte **reusable** del servicio
 - Posible distintas implementaciones concretas de esta interfaz
- Interfaz
 - Conjunto de operaciones que conforman un endpoint (bloque de **funcionalidad** del servicio)
 - **Patrón de intercambio** de mensajes para cada una
- Mensajes
 - Mensajes válidos a la entrada (AvailabilityMessage)
 - Mensajes que generará (ResponseMessage)
 - **datos** que se intercambian en cada mensaje con el servicio
- Vocabulario: Definición de **tipos de datos**
 - XML Schema Definition (XSD)
 - No es lo mismo

<pre><Fechas> <in>1/1/07</in> <total>17</total> </Fechas></pre>	<pre><FechaIn>1/1/07</FechaIn> <FechaOut>1/1/07</FechaOut></pre>
---	--



TIPOS

- Definir la colección de los datos y sus tipos que serán utilizados por el servicio web e intercambiados dentro de los mensajes
- Esta información es compartida entre cliente y servicio para que puedan ser interpretados correctamente a ambos lados.
 - El servicio web necesita acceder a la información que el cliente ha usado para codificar los datos y debe entender cómo decodificarlos.
 - El cliente del servicio web debe saber cómo codificar los datos para que el servicio no se queje.
- XML Schemas: tipos de datos no estándar
 - Por ejemplo, no es lo mismo definir una estructura persona (dni, nombre, apellidos, dirección, teléfono) que una estructura persona (dni, nombre, contacto), ..etc.
 - Ejemplo: Calculadora con Tipo de datos Complejos ($Re(z)$, $Im(z)$)
 - Todas las plataformas disponen de herramientas que generan luego las clases necesarias si no existiese el tipo en el lenguaje destino.



TIPOS: Descripción formal

```
<wsdl:types> ?  
  <wsdl:documentation .... /> ?  
  <xsd:schema .... /> *  
</wsdl:types>  
  
<xs:schema  
  xmlns:xs="http://www.w3.org/2001/XMLSchema"  
  targetNamespace="http://greath.example.com/2004/schemas/resSvc"  
  xmlns="http://greath.example.com/2004/schemas/resSvc">  
  
  <xs:element name="..." type="..." />  
  <xs:complexType name="...">  
    <xs:sequence>  
      <xs:element name="..." type="xs:date" /> *  
    </xs:sequence>  
  </xs:complexType>  
  
  <xs:element name="checkAvailabilityResponse" type="xs:double" />  
  
  <xs:element name="invalidDataError" type="xs:string" />  
  
  </xs:schema>  
</types>
```



TIPOS: Descripción formal

- <type> contiene la definición de tipos de datos
 - Definidos usando XML Schema Definition (XSD) u otro sistema similar
 - http://es.wikipedia.org/wiki/XML_Schema
- Dentro de la etiqueta se pueden definir elementos (<element>) o tipos complejos (<complexType>)
 - <element> nos permite definir información (similar a declarar una variable). Cada trozo de información, se define en su propio elemento.
 - Esa información formará luego parte de los mensajes que se intercambia.
 - A cada elemento le damos un nombre (name), para poder utilizarlo luego y especificamos su tipo (type) que puede ser definido o predefinido.
 - Los tipos complejos suelen ser secuencias de elementos simples.



TIPOS

- Casa Otilio espera recibir un mensaje en el que se le indique
 - Fecha de llegada: tipo date
 - Fecha de salida: tipo date
 - Tipo de habitación: tipo string
- Casa Otilio devuelve un mensaje indicando el precio total en euros (float)
- Describa la sección types que cree necesaria



TIPOS: Ejemplo

```
<types>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://greath.example.com/2004/schemas/resSvc">
    <xs:element name="checkAvailability" type="tCheckAvailability"/>
    <xs:complexType name="tCheckAvailability">
      <xs:sequence>
        <xs:element name="checkInDate" type="xs:date"/>
        <xs:element name="checkOutDate" type="xs:date"/>
        <xs:element name="roomType" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="checkAvailabilityResponse" type="xs:double"/>
    <xs:element name="invalidDataError" type="xs:string"/>
  </xs:schema>
</types>
```




TIPOS: Ejemplo

```
<wsdl:types>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="qualified"
    elementFormDefault="qualified" targetNamespace="http://Hotel/xsd">
    <xs:element name="GetDisponibilidad">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="fechaIn" type="xs:dateTime"/>
          ...
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="GetDisponibilidadResponse">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="return" nillable="true" type="ns:InfoDisponibilidad"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="InfoDisponibilidad" type="ns:InfoDisponibilidad"/>
    <xs:complexType name="InfoDisponibilidad">
      <xs:sequence/>
    </xs:complexType>
  </xs:schema>
</wsdl:types>
```



TIPOS: Tarea (10 minutos)

- Suponga un servicio web denominado Calculadora.
- Tendrá dos operaciones
 - `int suma (int, int)`
 - `int resta (int, int)`
- Defina la sección `types`

MENSAJES

- Describe los datos intercambiados entre el proveedor y el cliente del servicio web.
 - Tiene un nombre único para el conjunto de todos los mensajes
 - **Cada mensaje es un documento particionado (<part>)**
 - Cada parte se asocia con uno de los parámetros de la función a invocar, que se envían en el mensaje, o el valor de retorno.
 - Por ejemplo, la invocación de un procedimiento con dos parámetros, un entero y un real se puede definir como un mensaje con dos partes.
- Partes de los mensajes
 - Cada parte tiene un nombre (etiqueta name) que lo identifica dentro del mensaje
 - Se debe indicar cómo es la información que se intercambia
 - Bien con la etiqueta element que se corresponde con un elemento definido en la sección anterior
 - Bien con la etiqueta type, que indica el tipo concreto, el del parámetro que transmite

```
<message name="nmtoken"> *
```

```
    <part name="nmtoken"? element="qname" type="qname"? /> *
```

```
</message>
```

MENSAJES

■ mensajes. Ejemplo

```
<message name="SayHelloRequest">
  <part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse">
  <part name="greeting" type="xsd:string"/>
</message>
```

- Se definen dos elementos message.
- El primero representa un mensaje de solicitud, el segundo la respuesta.



MENSAJES

- **Tarea: Definir la sección mensajes de la calculadora anterior (10 minutos)**



OPERACIONES

- Equivalente a la signature del método en los lenguajes de programación
- Representa a cada uno de los métodos que se pueden invocar
 - Nombre del método
 - Parámetros de entrada (y tipo)
 - Parámetros de retorno (y tipo)
- Agrupa los mensajes de entrada y salida que se intercambian indicando si son input o output.



OPERACIONES

```
<operation name="xs:NCName"
    pattern="xs:anyURI" (in-only, out-only, in-out)
    style="list of xs:anyURI"?
    wsdlx:safe="xs:boolean"? >
    <input message="xs:NCName"?
        element="union of xs:QName, xs:Token"? >
    </input>*

    <output message="xs:NCName"?
        element="union of xs:QName, xs:Token"? >
    </output>*

</operation>*
```



INTERFAZ o PORTTYPE

- Define de forma abstracta un servicio web, las operaciones que puede llevar a cabo y los mensajes involucrados en cada operación

```
<wsdl:definitions .... >
  <wsdl:portType name="nmtoken">
    <wsdl:operation name="nmtoken" .... /> *
  </wsdl:portType>
</wsdl:definitions>
```

- El atributo **name** proporciona un nombre único entre todos los port-types definidos en el WSDL
- Para hacer referencia a una operación, se utiliza el atributo **name**
- Un endpoint soporta cuatro primitivas de transmisión, que se describen en el fichero WSDL: **One-way**, **Request-response**, **Solicit-response**, **Notification**.



One-way

- El cliente invoca un servicio enviando un único mensaje (1 mensaje, asíncrono)
- Ejemplo: Un mensaje para suscribirse a una lista de correos

```
<wsdl:operation name="setNombre">  
  <wsdl:input  
    message="setNombreMessage" />  
</wsdl:operation>
```



Notifications

- El servidor envía un único mensaje (1 mensaje, asíncrono)

```
<wsdl:operation name="nmtoken">  
  <wsdl:output name="nmtoken"?  
    message="qname" />  
</wsdl:operation>
```



Request-Response

- Se invoca al servidor y responde (2 mensajes, síncrono).
 - Para encapsular errores, se puede especificar también un elemento opcional fault.

```
<wsdl:operation name="nmtoken" parameterOrder="nmtokens">  
  <wsdl:input name="nmtoken"? message="qname"/>  
  <wsdl:output name="nmtoken"? message="qname"/>  
  <wsdl:fault name="nmtoken" message="qname"/>*</wsdl:operation>
```

Solicit-Response

- El servidor invoca y espera (2 mensajes, síncrono)

```
<wsdl:operation name="nmtoken"
    parameterOrder="nmtokens">
    <wsdl:output name="nmtoken"?
        message="qname" />
    <wsdl:input name="nmtoken"?
        message="qname" />
    <wsdl:fault name="nmtoken"
        message="qname" />*
</wsdl:operation>
```



INTERFAZ o PORTTYPE (ejemplo)

```
<portType name = "UsuarioRegistrado" >
  <operation name="opCheckAvailability"
    pattern="http://www.w3.org/ns/wsd1/in-out"
    style="http://www.w3.org/ns/wsd1/style/iri"
    wsdlx:safe = "true">
    <input messageLabel="In"
      element="ghns:checkAvailability" />
    <output messageLabel="Out"
      element="ghns:checkAvailabilityResponse" />
    <fault ref="tns:invalidDataFault" messageLabel="Out"/>
  </operation>
</ portType>
```

INTERFAZ o PORTTYPE

- Tarea (10 min): ¿Signatura del siguiente wsdl?

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:tns="http://example"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example">
  <wsdl:types>
    <schema elementFormDefault="qualified" targetNamespace="http://example"
      xmlns="http://www.w3.org/2001/XMLSchema" xmlns:apachesoap="http://xml.apache.org/xml-soap"
      xmlns:tns="http://example" xmlns:intf="http://example" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" >
      <element name="pasame">
        <complexType> <sequence>
          <element name="numero" type="xsd:string"/>
          <element name="saldo" type="xsd:int"/>
        </sequence>
      </complexType>
    </element>
    <element name="pasameResponse">
      <complexType> <sequence>
        <element name="saldo" type="xsd:int"/>
      </sequence>
    </complexType>
  </element>
</wsdl:types>
</definitions>
```



INTERFAZ o PORTTYPE

```
<element name="InsufficientFundFault"><complexType><sequence>
  <element name="numero" type="xsd:string"/>
  <element name="saldoActual" type="xsd:int"/>
  <element name="requestedSaldo" type="xsd:int"/>
</sequence></complexType>
</element>
</schema>
</wsdl:types>
```

INTERFAZ o PORTTYPE

- Tarea (10 min): ¿Signatura del siguiente wsdl?

```
<wsdl:message name="pasameRequest">
  <wsdl:part name="parameters" element="tns:pasame"/>
</wsdl:message>
<wsdl:message name="pasameResponse">
  <wsdl:part name="return" element="tns:pasameResponse"/>
</wsdl:message>
<wsdl:message name="InsufficientFundFaultMessage">
  <wsdl:part name="fault" element="tns:InsufficientFundFault"/>
</wsdl:message>
<wsdl:portType name="PasameSaldo">
  <wsdl:operation name="pasame">
    <wsdl:input name="pasameRequest" message="tns:pasameRequest"/>
    <wsdl:output name="pasameResponse" message="tns:pasameResponse"/>
    <wsdl:fault name="InsufficientFundException"
message="tns:InsufficientFundFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```




INTERFAZ o PORTTYPE

- Tarea: Calculadora (10 min)
- Si le resulta fácil, ponga también la operación de división



Especificación WSDL 1.2: Parte Concreta

- Types, Messages y PortType definen la parte abstracta, **reusable** del WSDL
- Pero no suministran detalles del protocolo de transporte, el formato del mensaje o la codificación de datos para todas las operaciones y mensajes definidos en una interfaz, especificando **cómo** se van a enviar/recibir los mensajes.
- Para definir (concretar) una instancia de servicio real, se tiene que definir :
 - El conjunto exacto de interfaces que implementa y en qué direcciones están accesibles (etiqueta port)
 - Los protocolos de transporte usados y la codificación de los datos.(etiqueta binding)
- http://www.microsoft.com/spanish/msdn/articulos/archivo/091101/voices/wsdlexplained.asp#wsdlexplained_topic08
- http://www.microsoft.com/spanish/msdn/articulos/archivo/090201/voices/wsdlexplained.asp#_bindings

BINDING

- El término **enlace(binding)** hace referencia al proceso que asocia a una entidad abstracta (mensaje, operación, portType), información sobre un protocolo o formato de datos
 - Especifica cómo mapear cada operación abstracta y sus mensajes en el formato de mensajería que se decida usar.

```
<wsdl:binding name="nmtoken" type="qname"> *
<!-- extensibility element (1) --> *
  <wsdl:operation name="nmtoken"> *
    <!-- extensibility element (2) --> *
    <wsdl:input name="nmtoken"? > ?
      <!-- extensibility element (3) -->
    </wsdl:input>
    <wsdl:output name="nmtoken"? > ?
      <!-- extensibility element (4) --> *
    </wsdl:output>
    <wsdl:fault name="nmtoken"> *
      <!-- extensibility element (5) --> *
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!--Optional header information goes here. -->
    <To>Scott</To>
    <From>Suzanne</From>
  </soap:Header>
  <soap:Body>
    <!--Message goes here. -->
    Please pick up some milk on your way home from work.
  </soap:Body>
</soap:Envelope>
```



BINDING

- Imaginemos un servicio web con una operación `getStockPrice`. Dos parámetros: símbolo de la compañía y fecha para la que dar el precio.
 - Podemos imaginar el tipo de documento XML que se transmitirá pero....NO SABEMOS CÓMO FORMATEAR ESE MENSAJE PARA INVOCAR ESAS OPERACIONES
 - Supongamos que el servicio web usa SOAP como formato de cable
 - ¿ Problema?: La especificación WSDL es **independiente** de la especificación SOAP
 - No hay etiquetas wsdl para formatear el mensaje soap
 - Pero necesitamos un medio para describir cómo formatear las peticiones en el envelope SOAP.
 - Por ejemplo, si no se especifica, ¿cómo sabes si los parámetros van en la cabecera o en el cuerpo del mensaje?



Binding extensions

- Solución: WSDL define un **mecanismo de extensibilidad** que permite extender las etiquetas wsdl <binding> con elementos de diferentes espacios de nombres XML
- WSDL admite elementos (etiquetas) que representan una tecnología específica (denominados **elementos de extensibilidad** o binding extensions) bajo varios elementos definidos por WSDL (como si fuesen sus hijos).
 - Estos binding extensions se utilizan normalmente con el fin de especificar información de enlace para un protocolo o formato de mensaje determinado, pero no se limitan a dicho uso.
 - Los elementos de extensibilidad DEBEN (MUST) utilizar un espacio de nombres de XML distinto al de WSDL.

BINDING o INTERFACEBINDING

```
<binding...>
  <soap:binding style="rpc|document" transport="uri">
    <operation...>
      <soap:operation soapAction="uri"? style="rpc|document"?>?
      <input>
        <soap:body parts="nmtokens"? use="literal|encoded" encodingStyle="uri-list"? namespace="uri"?>
        <soap:header element="qname" fault="qname"?>*>
      </input>
      <output>
        <soap:body parts="nmtokens"? use="literal|encoded" encodingStyle="uri-list"? namespace="uri"?>
        <soap:header element="qname" fault="qname"?>*>
      </output>
      <fault>*>
        <soap:fault name="nmtoken" use="literal|encoded" encodingStyle="uri-list"? namespace="uri"?>
      </fault>
    </operation>
  </binding>

  <port...>    <soap:address location="uri"/>    </port>
```



Binding extensions

- Como lo más habitual es enviar SOAP sobre HTTP existe una extensión para SOAP (binding extension) que se describe en la especificación.
 - Aumenta las etiquetas con aquellas que nos van a permitir indicar cómo componer un mensaje SOAP y enviarlo sobre HTTP de la forma más fácil posible
- La especificación WSDL describe también extensiones para HTTP GET/POST (que usa HTTP sin SOAP codificando los datos en la URL), MIME.
 - Las extensiones para HTTP y MIME permiten definir los requerimientos para comunicar con aplicaciones web que pueden (o no) devolver documentos XML



Extensión para SOAP

- La extensión SOAP se usa para definir un servicio que soporta el protocolo SOAP
 - Permite especificar detalles específicos de SOAP, como la cabecera de SOAP, los estilos de codificación de SOAP, y SOAPAction de la cabecera HTTP.
 - De esta forma, no tendremos que construir nosotros el mensaje “a pelo” en el cliente. Las herramientas generarán un stub que actuará según lo que se indique en el fichero wsdl
- El enlace SOAP extiende WSDL con los siguientes elementos de extensión
 - Estilo de la invocación
 - SOAPAction en la cabecera
 - Apariencia de los mensajes de entrada y salida
 - Formateado de la cabecera del mensaje SOAP
 - Formateado de los elementos Fault y HeaderFault del mensaje SOAP.

Extensión para SOAP

```
<soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
```

- Indica que todas las operaciones se definen como mensajes SOAP (sobre, encabezado, cuerpo)
- El atributo style indica cómo traducir la información a un mensaje SOAP (dónde colocar cada cosa): Valores posibles RPC y document (valor por defecto)
- Tiene un atributo transport (habitualmente <http://schemas.xmlsoap.org/soap/http>) indica que se utiliza SOAP sobre HTTP como protocolo de transporte, mientras que si se desea SOAP sobre SMTP, se usa <http://schemas.xmlsoap.org/soap/smtp>
- Ejemplo: `<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>`

Extensión para SOAP

□ InterfaceBindings

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="sayHello">
    <soap:operation soapAction="sayHello"/>
    <input>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:examples:helloservice" use="encoded"/>
    </input>
    <output>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:examples:helloservice" use="encoded"/>
    </output>
  </operation> </binding>
```



RPC/Document, literal/encoded

- Which style of WSDL should I use? (<http://www-106.ibm.com/developerworks/webservices/library/ws-whichwsdl>)
- Reap the benefits of document style Web services (<http://www-106.ibm.com/developerworks/webservices/library/ws-docstyle.html>)
- Discover SOAP encoding's impact on Web service performance (<http://www-106.ibm.com/developerworks/webservices/library/ws-soapenc>)
- Los dos aspectos que determinan cómo se construyen la cabecera y el cuerpo de un mensaje SOAP son el estilo de interacción y las reglas de codificación.
- Estilo de interacción
 - Determina cómo construir el mensaje (RPC, Document)
- Reglas de codificación
 - Determina cómo codificar los parámetros (literal, encoded)



Document

- Los mensajes SOAP se usan para transportar documentos XML de una aplicación a otra
- La característica principal es que, cada parte del mensaje intercambiado (que enlaza con los elementos `<soap:binding>`) se etiqueta con uno de los `<element>` definidos en el schema
 - Esto significa que el mensaje se puede describir completamente usando la declaración de `<element>` en el `<schema>`
- Restricciones. La más importante:
 - Los mensajes input y output sólo pueden tener un elemento `<part>`
 - En el mensaje de solicitud ese `<part>` se refiere a un element CON EL MISMO NOMBRE QUE DEMOS A LA OPERACIÓN*ii*



Document

```
<types>
  <schema ...>
    <element name="addNumbers">
      <xsd:complexType>
        <sequence>
          <element name="number1" type="xsd:int"/>
          <element name="number2" type="xsd:int"/>
        </sequence>
      </xsd:complexType>
    </element>
    <element name="addNumbersResponse"> ... </element>
  </schema>
</types>

<message name="addNumbersRequest">
  <part name="param" element="ns1:addNumbers"/>
</message>
<message name="addNumbersResponse"> ... </message>

<portType name="AddNumbersPortType">
  <operation name="addNumbers" parameterOrder="param">
    <input message="tns:addNumbersRequest"/>
    <output message="tns:addNumbersResponse"/>
    <fault name="error" message="tns:addNumbersFault"/>
  </operation>
</portType>
```



Document

```
<binding name="AddNumbersBinding" type="tns:AddNumbersPortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="addNumbers">
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="error">
        <soap:fault name="error" use="literal"/>
      </fault>
    </operation>
  </binding>
```

Document

- Signatura generada

- public int addNumbers(int number1, int number2) throws AddNumbersFault, java.rmi.RemoteException;

- Mensaje SOAP

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

```
<env:Envelope
```

```
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
```

```
  xmlns:ns0="http://wombat.org/types">
```

```
    <env:Body>
```

```
      <ns0:addNumbers>
```

```
        <number1>1</number1>
```

```
        <number2>2</number2>
```

```
      </ns0:addNumbers>
```

```
    </env:Body>
```

```
</env:Envelope>
```



RPC

- El cuerpo del mensaje SOAP contiene la llamada y los parámetros
 - La llamada se etiqueta con el nombre de la operación
 - Los parámetros con cada uno de los elementos <part> del mensaje que tienen que haber sido definidos usando el elemento **type**
- El cuerpo de la respuesta contiene resultado y parámetros de salida
- Ambas aplicaciones tienen que ponerse de acuerdo en la signatura
- Lectura: “Reap the benefits of document style web services”



RPC

```
<message name="addNumbersRequest">
  <part name="number1" type="xsd:int"/>
  <part name="number2" type="xsd:int"/>
</message>
<message name="addNumbersResponse">
  <part name="return" type="xsd:int"/>
</message>
<portType name="AddNumbersPortType">
  <operation name="addNumbers" parameterOrder="number1 number2">
    <input message="tns:addNumbersRequest"/>
    <output message="tns:addNumbersResponse"/>
  </operation>
</portType>
<binding name="AddNumbersBinding" type="tns:AddNumbersPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="addNumbers">
    <input> <soap:body use="literal" namespace="..."/> </input>
    <output>
      <soap:body use="literal" namespace="..."/> </output>
    </operation>
  </binding>
```

RPC

- Signatura generada: `public int addNumbers(int number1, int number2)`
`throws AddNumbersFault, java.rmi.RemoteException;`

- Mensaje enviado

```
<env:Envelope
  xmlns:env=http://schemas.xmlsoap.org/soap/envelope/
  .....
<env:Body>
  <ns1:addNumbers>
    <number1>1</number1>
    <number1>2</number1>
  </ns1:addNumbers>
</env:Body>
<env:Envelope>
```



Literal/encoded

- Las reglas de codificación determinan cómo se representa en XML una estructura de datos particular y cómo se transforman en XML un tipo de datos Java
- Para que cliente y servicio puedan interoperar tienen que ponerse de acuerdo en cómo codificar los datos.
- "Encoded" indica que se han codificado aplicando para la codificación unas reglas determinadas definidas en el atributo "encodingStyle"
- SOAP 1.2 define una forma particular de codificación llamada SOAP encoding.
 - Define cómo estructuras de datos incluyendo tipos básicos (integer, ..) o tipos complejos (arrays, estructuras) se serializan en XML
- "Literal" significa que el mensaje SOAP resultante contiene datos formateados exactamente como se especifica en las definiciones abstractas (XML Schema).



Formateado de los mensajes SOAP

- Formateado de SOAPAction en la petición HTTP
- Formateado de la cabecera
- Formateado del cuerpo



SOAPAction en la cabecera

- Para cada operación expuesta en la interfaz que estamos concretando, habrá que incluir la etiqueta `<soap:operation soapAction="nombre"/>`
 - El valor que se asigne a SOAPAction será el que aparezca en el encabezado HTTP.
 - También habrá que indicar si se transmite en la cabecera o el cuerpo del mensaje (soap:body, soap:header, soap:fault)

Formateado de la cabecera del mensaje SOAP *<soap:header>*

- Indica qué se pone y cómo en la cabecera
- Sus atributos son similares a los de *<soap:body ...>*

```
<operation name="OpName">
  <soap:operation
    soapAction="http://...../registration">
    <input>
      <soap:header

message="StockAvailableRegistrationRequest"
      part="expiration" use="encoded"
      namespace="http://.."
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
/>
```

Apariencia de los mensajes de entrada y salida `<soap:body>`

- El mensaje debe aparecer como parte del cuerpo del mensaje SOAP y permite describir, exactamente, cómo aparecen los mensaje de entrada (input) en el cuerpo del mensaje SOAP

```
<input>
  <soap:body use="encoded"
    namespace="http://www.skatestown.com/services/PriceCheck"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</input>
```

- **Parámetro Use:** Indica qué reglas de codificación se usan para serializar las distintas partes de un mensaje en XML (literal, encoded)
- **Parámetro Namespace:** Cada cuerpo de mensaje SOAP puede tener su propio espacio de nombre para evitar conflictos de nomenclatura. El identificador URI especificado en este atributo se utiliza literalmente en el mensaje SOAP resultante.
- **Parámetro EncodingStyle:** En la codificación SOAP, debería tener el valor URI "http://schemas.xmlsoap.org/soap/encoding".



Binding extensions

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns3:checkPrice
      xmlns:ns3="http://www.skatestown.com/services/PriceCheck">
      <sku xsi:type="xsd:string">947-TI
      </sku>
    </ns3:checkPrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Formateado de los elementos Fault del mensaje SOAP

- La extensión soap:fault describe cómo codificar, en un mensaje SOAP, los elementos que se transmiten en caso de fallo

```
<soap:binding
  transport="http://schemas.xmlsoap.org/soap/http"
  style="document" />
<operation name="addNumbers">
  <soap:operation soapAction="" />
  <input> <soap:body use="literal" /> </input>
  <output> <soap:body use="literal" /> </output>
  <fault name="addNumbersFault">
    <soap:fault name="addNumbersFault" use="literal" />
  </fault>
</operation>
```

- Tiene los mismos atributos que la extensión soap:body
- El mensaje fault debe tener una única parte

Port, EndPoint

- Combinan la información de los binding con direcciones de red (especificados por URI) en la que se puede acceder a la implementación del tipo de puerto.

```
<wsdl:definitions .... >
  <wsdl:service .... > *
    <wsdl:port name="nmtoken" binding="qname" > *
      <!-- extensibility element (1) -->
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

- El elemento port tiene dos atributos, **name** y **binding**
- La parte extensible (1) se suele usar para especificar la información de direccionamiento del puerto.
 - ☐ Etiqueta soap:address, atributo location
 - ☐ No debe ser más de una
 - ☐ No debe haber más información que esta



Port, EndPoint

■ Ejemplo:

```
<service name="Hello_Service">
  <documentation>
    WSDL File for HelloService
  </documentation>
  <port binding="tns:Hello_Binding"
    name="Hello_Port">
    <soap:address
      location="http://www.examples.com/SayHello/">
    </port>
</service>
```



Service

- Agrupaciones lógicas de puertos. Un servicio específico podría estar accesible en diferentes direcciones (p.e. URIs de diferentes máquinas)
 - Normalmente un servicio agrupa puertos relacionados, disponibles con frecuencia en una misma dirección.
 - Otra agrupación posible es utilizar diferentes **ports** que representan diferentes **bindings** del mismo tipo de puerto.
 - Permiten, en definitiva, que la misma funcionalidad sea accesible en diferentes estilos de interacción y con diferentes protocolos de transporte
 - Incluye una etiqueta *documentation* para proporcionar una descripción humanamente legible



Ejemplo

```
<service name="Hello_Service">
  <documentation>WSDL File for HelloService</documentation>
  <port binding="tns:Hello_Binding" name="Hello_Port">
    <soap:address location="http://www.examples.com/SayHello/"> </port>
  </service>
  <binding name="Hello_Binding" type="tns:Hello_PortType">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice" use="encoded"/> </input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice" use="encoded"/> </output>
      </operation>
    </binding>
```

Especificación WSDL 1.2:

Documento completo

- **<definitions>**: Raíz del wsdl, contiene la definición de uno o más servicios (habitualmente, uno).

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri"?  
    Namespace=URI...>
```

- **Atributos**

- **targetNamespace**: Define el espacio de nombre lógico para calificar la información acerca del servicio. Los nombres que demos a interfaces, bindings y servicios en nuestro documento se asociarán con este espacio de nombres y así se distinguirán de nombres similares en otros espacios de nombres
 - Dado que los ficheros WSDL pueden importar otros ficheros WSDL, siempre existe la posibilidad de una colisión de nombres
 - targetNamespace se asocia con un URL único para el fichero WSDL, habitualmente, su nombre
- **xmlns:tns**: Si aparece, debe tener el mismo valor que targetNamespace. En el resto del documento podremos usar el prefijo tns para calificar los nombres de los mensajes, tipos, puertos, enlaces operaciones y servicios que definamos.
- Podemos definir más abreviaturas para otros espacios de nombres utilizados en el resto del documento
- **xmlns:soap** and **xmlns:xsd**: Definiciones estándar de espacios de nombres que se usan ampliamente en el documento WSDL para especificar información específica de SOAP y tipos de datos, respectivamente.
- **xmlns**: Espacio de nombres por defecto para un documento WSDL. Todas las etiquetas WSDL como <service>, <endpoint>, <message>, residen en este espacio de nombres

- El elemento <schema> tiene su propio atributo targetNamespace y todos los nombres definidos en este elemento <schema> pertenecen a este espacio de nombre no al principal.

Especificación WSDL 1.2:

Documento completo

- **<definitions>**: Es la raíz del documento wsdl y contiene la definición de uno o más servicios (en la mayoría de los casos, uno).

```
<wsdl:definitions name="nmtoken" ?  
  targetNamespace="uri" ? Namespace=URI... >
```

- Atributos


- targetNamespace: Define el espacio de nombre lógico para calificar la información acerca del servicio. Los nombres que demos a interfaces, bindings y servicios en nuestro documento se asociarán con este espacio de nombres y así se distinguirán de nombres similares en otros espacios de nombres
 - Dado que los ficheros WSDL pueden importar otros ficheros WSDL, siempre existe la posibilidad de un colisión de nombres
 - targetNamespace se asocia con un URL único para el fichero WSDL, habitualmente, su nombre
- xmlns:tns: Si aparece, debe tener el mismo valor que targetNamespace. En el resto del documento podremos usar el prefijo tns para calificar los nombres de los mensajes, tipos, puertos, enlaces operaciones y servicios que definamos.
- xmlns:soap y xmlns:xsd: Definiciones estándar de espacios de nombres que se usan ampliamente en el documento WSDL para especificar información específica de SOAP y tipos de datos, respectivamente.
- xmlns: Espacio de nombres por defecto para un documento WSDL. Todas las etiquetas WSDL como <service>, <endpoint>, <message>, residen en este espacio de nombres

- El elemento <schema> tiene su propio atributo targetNamespace y todos los nombres definidos en este elemento <schema> pertenecen a este espacio de nombre no al principal.

Especificación WSDL 1.2:

Documento completo

Prefijo	URI del espacio de nombres	Definición
wsdl	http://schemas.xmlsoap.org/wsdl/	Espacio de nombres WSDL para el framework WSDL
soap	http://schemas.xmlsoap.org/wsdl/soap/	Espacio de nombres WSDL para el enlace WSDL SOAP
http	http://schemas.xmlsoap.org/wsdl/http/	Espacio de nombres WSDL para en enlace WSDL HTTP
mime	http://schemas.xmlsoap.org/wsdl/mime/	Espacio de nombres WSDL para el enlace WSDL MIME
soapenc	http://schemas.xmlsoap.org/soap/encoding/	Espacio de nombres para la codificación
soapenv	http://schemas.xmlsoap.org/soap/envelope/	Envelope namespace as defined by SOAP 1.1 [8].
xsd	http://www.w3.org/2000/10/XMLSchema	Espacio de nombres de Schema (para poder definir tipos complejos utilizando esquema)
tns	(varios)	El prefijo tns es una abreviatura al “espacio de nombres actual”, usado como convención para referirse a las definiciones hechas en el documento actual
otros	(varios)	El resto de los prefijos son de casos concretos. URI's que comienzan con “http://example.com” representan URIs dependientes de la aplicación



Especificación WSDL 1.2:

Documento completo

```
<definitions name="FooSample"
  targetNamespace="http://tempuri.org/wsdl/"
  xmlns:wsdlns="http://tempuri.org/wsdl/"
  xmlns:typens="http://tempuri.org/xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:stk="http://schemas.microsoft.com/soap-toolkit/wsdl-
extension"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema targetNamespace="http://tempuri.org/xsd"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      elementFormDefault="qualified" >
      </schema>
    </types>
```



WSDL (Web Services Description Language) Construcción de servicios web

Lourdes Tajés Martínez
(tajes@uniovi.es)



Desarrollo de una aplicación

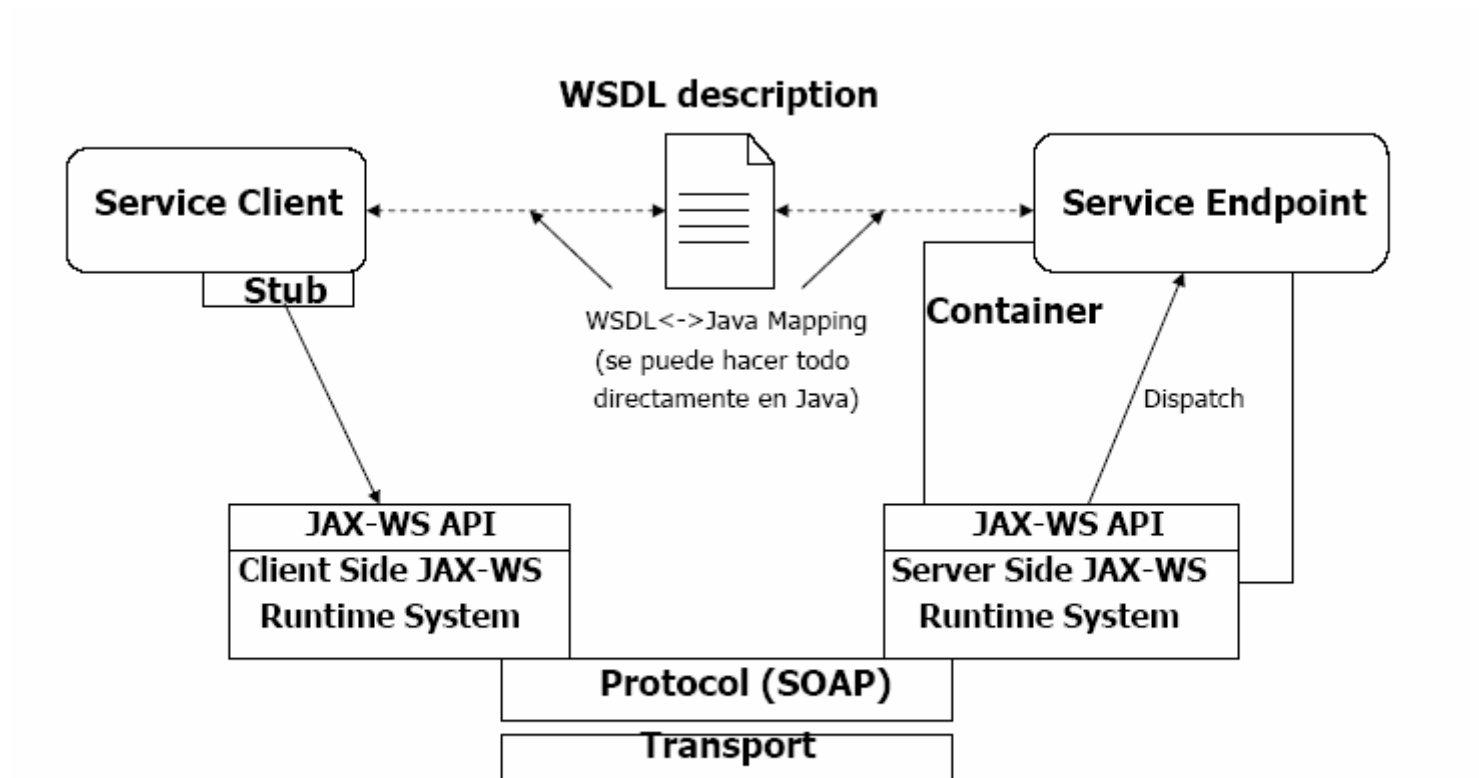
- <http://edocs.bea.com/wls/docs100/webserv/jaxws.html>
- <http://open.ionac.com/docs/framework/2.0/jaxws/jaxws.pdf>
- **Servicio**
 - **Construcción del servidor**
 - Interfaz java de servidor
 - Implementación de servidor
 - Compilación java
 - Generar artefactos para WS (apt, wsgen)
 - **Desplegar servicio**
- **Cliente**
 - wsimport: Importar información de WS para el cliente
 - Codificar el cliente
 - Ejecutar el cliente



Desarrollo de una aplicación

- WSDL funciona como contrato entre cliente y servicio
- Se utiliza de manera **similar a IDL**
 - Existen “**traducciones**” de WSDL a diferentes lenguajes de programación
- Los **compiladores WSDL** generan “stub” y “skeleton” para clientes y servicios SOAP en los diferentes lenguajes
- Ej.: Traducciones definidas por Sun Microsystems
 - WSDL -> Java (wsimport)
 - Java -> WSDL (wsген, apt)


Desarrollo de una aplicación





WSDL Y JAX-WS

- Existen dos herramientas fundamentales: wsimport y wsgen o apt
- wsimport
 - Genera, a partir de un fichero wsdl correcto, gran parte del código del servicio web que se corresponde con esa descripción
 - Si hablamos del cliente, también genera los ficheros necesarios (stub) para actuar como intermediarios entre el cliente y el servicio y facilitar la escritura de los primeros (por ejemplo, no tienen que construir el mensaje “a pedal”, el stub lo hará)
- wsgen o apt
 - Puedes utilizar esta herramienta para generar el fichero wsdl que lo describe y publicarlo a potenciales clientes a partir de una implementación java **anotada**



Modelos de desarrollo del servicios web: Comenzar por la clase


- Se ha de proporcionar una clase que implemente un endpoint válido
 - Debe llevar una `javax.jws.WebService` annotation (JSR 181).
 - Puede extender `java.rmi.Remote` (directa o indirectamente)
 - Sus métodos pueden llevar una `javax.jws.WebMethod` annotation
 - Todos los parámetros de sus métodos y tipos de retorno deben ser compatibles con JAXB 2.0 Java to XML Schema mapping
 - Los parámetros de los métodos y tipos de retorno no deben implementar la interfaz `java.rmi.Remote` ni directa ni indirectamente.

Modelos de desarrollo del servicios web: Comenzar por la clase

- Ejemplo que muestra una clase que implementa un endpoint válido ([AddNumbersImpl.java](#))

```
package fromjava.server;
import javax.xml.ws.WebService;
@WebService
public class AddNumbersImpl {
    /**
     * @param number1
     * @param number2
     * @return The sum
     * @throws AddNumbersException
     * if any of the numbers to be added is negative.
     */
    public int addNumbers(int number1, int number2) throws AddNumbersException {
        if (number1 < 0 || number2 < 0) {
            throw new AddNumbersException("Negative number cant be added!",
                "Numbers: " + number1 + ", " + number2);
        }
        return number1 + number2;
    }
}
```

- Aunque no se requiere, es común proporcionar una SEI.



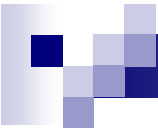
Modelos de desarrollo del servicios web: Comenzar por la clase

- Herramientas: wsgen o apt (recomendada)
 - apt (<http://java.sun.com/webservices/docs/2.0/jaxws/apt.html>): Es necesario el código fuente
- Los **ficheros generados** incluyen clases para representar los mensajes, codificar los parámetros y las excepciones
 - AddNumbers.(java|class) : Representa la solicitud y sus parámetros
 - AddNumbers_Exception.(java|class): Se encargan de representar la excepción
 - AddNumbersResponse.(java|class): Se encargan de procesar la respuesta



¿ Cómo vamos?

- Estoy cansada de hablar...
 - ☐ Ir a “Desarrollo del cliente” (91)
 - ☐ Ejercicios
- Es casi imposible que no esté cansada de hablar pero, por si acaso...
 - ☐ Seguir



Modelos de desarrollo del servicios web: Comenzar por el WSDL

- Describir el servicio en forma de mensajes intercambiados, protocolo de mensajería y transporte y dirección de servicio.
- A partir del wsdl, habrá que generar las clases necesarias para desarrollar el servicio y el cliente utilizando **wsimport**
 - SEI (Service Endpoint Interface)
 - Se corresponde con la interfaz lógica, definida en el elemento <wsdl:portType>
 - Tiene anotaciones que se podrán usar para enlazar dinámicamente el servicio
 - Podemos usarla para generar un WSDL y un fichero schema (que podrían ser diferentes a los de partida).
 - Cualquier tipo complejo definido en el WSDL (schema) se corresponde con clases java siguiendo la correspondencia establecida en la especificación Java Architecture for XML Binding (JAXB)
 - Skeleton (clase que implementa el SEI)
 - Habrá que terminar de codificarla
 - Se corresponde con el elemento portType y tendrá como nombre portTypeNameImpl.java
 - Es necesario anotarla con @WebService, con un elemento endpointInterface que especifique el nombre de la clase SEI
 - Una clase java que se corresponde con <wsdl:service> de nombre serviceNameService.java que será utilizada por los consumidores para acceder a la implementación del servicio



Clases generadas PARA EL PROVEEDOR del servicio


Fichero	Descripción
portTypeName.java	El SEI. Contiene la interfaz que debe implementar tu servicio. Tiene anotaciones que se podrán usar para enlazar dinámicamente el servicio o para serializar/deserializar los parámetros. Podemos usarla para generar un WSDL y un fichero schema (que podrían ser diferentes a los de partida).
serviceName.java	El endpoint. Contiene la clase java que los consumidores usarán para hacer solicitudes.
portTypeNameImpl.java	El skeleton. Debes codificar la clase para construir el proveedor del servicio.
portTypeNameServer.java	Un servidor básico que te permite desplegar tu servicio como un proceso stand alone. No lo vamos a utilizar
Nombre_Exception.java	Clases para las excepciones definidas por el usuario (throwable). El nombre coincide con el de la etiqueta <fault>_Exception
nombre.java	Clases para todos los tipos definidos en el wsdl

Modelos de desarrollo de servicios web: Comenzar por el WSDL

■ Ejemplo: AddNumbers.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
  <definitions name="AddNumbers" targetNamespace=http://duke.example.org xmlns:tns=http://duke.example.org
    xmlns=http://schemas.xmlsoap.org/wsdl/ xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/>


    <types>
      <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
        targetNamespace="http://duke.example.org">
        <complexType name="addNumbersResponse">
          <sequence> <element name="return" type="xsd:int" /> </sequence>
        </complexType>
        <element name="addNumbersResponse" type="tns:addNumbersResponse" />
        <complexType name="addNumbers">
          <sequence>
            <element name="arg0" type="xsd:int" />
            <element name="arg1" type="xsd:int" />
          </sequence>
        </complexType>
        <element name="addNumbers" type="tns:addNumbers" />
        <element name="AddNumbersFault" type="tns:AddNumbersFault" />
        <complexType name="AddNumbersFault">
          <sequence>
            <element name="faultInfo" type="xsd:string" />
            <element name="message" type="xsd:string" />
          </sequence>
        </complexType>
        <complexType name="oneWayInt">
          <sequence>
            <element name="arg0" type="xsd:int" />
          </sequence>
        </complexType>
        <element name="oneWayInt" type="tns:oneWayInt" />
      </xsd:schema>
    </types>
```



Modelos de desarrollo del servicios web: Comenzar por el WSDL

```
<message name="addNumbers">
  <part name="parameters" element="tns:addNumbers" />      </message>
<message name="addNumbersResponse">
  <part name="result" element="tns:addNumbersResponse" /></message>
<message name="addNumbersFault">
  <part name="AddNumbersFault" element="tns:AddNumbersFault"/> </message>
<message name="oneWayInt">
  <part name="parameters" element="tns:oneWayInt" />      </message>


<portType name="AddNumbersPortType">
  <operation name="addNumbers">
    <input message="tns:addNumbers" name="add"/>
    <output message="tns:addNumbersResponse" name="addResponse"/>
    <fault name="addNumbersFault" message="tns:addNumbersFault"/>
  </operation>
  <operation name="oneWayInt"> <input message="tns:oneWayInt"/>
  </operation>
</portType>
```



Modelos de desarrollo del servicios web: Comenzar por el WSDL

```
<binding name="AddNumbersBinding" type="tns:AddNumbersPortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="addNumbers">
    <soap:operation soapAction="" />
    <input>    <soap:body use="literal" />          </input>
    <output>   <soap:body use="literal" />          </output>
    <fault name="addNumbersFault"> <soap:fault
      name="addNumbersFault" use="literal" /> </fault>
    </operation>
  </binding>

<operation name="oneWayInt">
  <soap:operation soapAction="" />
  <input>    <soap:body use="literal" />          </input>
</operation>
</binding>
```



Modelos de desarrollo del servicios web: Comenzar por el WSDL

```
<service name="AddNumbersService">  
  <port name="AddNumbersPort" binding="tns:AddNumbersBinding">  
    <soap:address location="REPLACE_WITH_ACTUAL_URL" />  
  </port>  
</service>  
  
</definitions>
```


Modelos de desarrollo de servicios web: Comenzar por el WSDL

- Ejemplo: SEI generada desde el wsdl

```
package fromwsdl.server;
```

```
@javax.jws.WebService(  
    name="AddNumbersPortType", // el nombre que venga en el wsdl  
    serviceName="AddNumbersService", // idem  
    targetNamespace=http://duke.org // idem  
)  
@javax.jws.soap.SOAPBinding(  
    style=javax.jws.soap.SOAPBinding.Style.DOCUMENT,  
    use=javax.jws.soap.SOAPBinding.Use.LITERAL,  
    parameterStyle=javax.jws.soap.SOAPBinding.ParameterStyle.WRAPPED)  
public interface AddNumbersPortType extends java.rmi.Remote {  
    @javax.jws.WebMethod(operationName="addNumbers") // idem  
    @javax.jws.WebResult(name="return")  
    public int addNumbers(  
        @javax.jws.WebParam(name="arg0")  
        int arg0,  
        @javax.jws.WebParam(name="arg1")  
        int arg1) throws fromwsdl.server.AddNumbersFault_Exception,  
        java.rmi.RemoteException;  
}
```

Modelos de desarrollo del servicios web: Comenzar por el WSDL

□ Ejemplo: Implementación de SEI

```
package fromwsdl.server;
```

```
@WebService(endpointInterface="fromwsdl.server.AddNumbersPortType")
public class AddNumbersImpl implements AddNumbersPortType {
    /**
     * @param number1
     * @param number2
     * @return The sum
     * @throws AddNumbersException
     * if any of the numbers to be added is negative.
     */

    public int addNumbers(int number1, int number2)
    throws AddNumbersFault_Exception {
        ...
    }
}
```



Desarrollo del cliente

- Generar ficheros a partir del wsdl: Herramienta wsimport
 - Genera el localizador del servicio, clase llamada serviceName.java. Se corresponde con la etiqueta <service> del wsdl. Contiene un método getPort para endpoint instalado para al servicio y definido en el wsdl
 - Una clase que realiza las tareas de proxy, encapsulando las operaciones para invocar métodos. Se corresponde con <wsdl:portType> y se llama portTypeName.java.
 - Clases para los tipos y excepciones



Desarrollo del cliente

- Una aplicación cliente puede acceder al servicio web remoto de dos maneras: proxy y dispatch
- **Proxy**
 - https://www.ibm.com/developerworks/websphere/library/techarticles/0709_thaker/0709_thaker.html
 - Los proxies para un servicio web se crean a partir del SEI generado y su implementación (o del WSDL)
 - Una vez que los proxies están creados, el cliente puede invocar los métodos de estos proxies como si fuesen una implementación local de SEI



Desarrollo del cliente

■ Codificación del cliente:

- Primero creamos el *localizador* del servicio (con AddNumbersService)
- A partir de éste obtenemos un proxy (de tipo XXXPortType)
- Invocamos al método local, que encapsula la invocación de la operación del Servicio Web
 - Se llaman igual, no nos daremos cuenta.



Ejemplo

```
package fromwsdl.client;

public class AddNumbersClient {
    public static void main (String[] args) {
        //          AddNumbersPortType port = new
        // AddNumbersService().getAddNumbersPort ();

        // Crea un service factory y carga la SEI
        AddNumbersService factoria= new AddNumbersService();

        // Obtiene una instancia del stub generado
        AddNumbersPortType port= factoria.getAddNumbersPort();
        System.out.println("Invoking one-way operation. Nothing is
returned from service.\n");

        // invoca el método remoto
        port.oneWayInt(10);
    }
}
```



Nuestra plataforma

- Java Platform, Standard Edition 6 (Java SE 6)
 - Soporte para Java API for XML Web Services (JAX-WS), versión 2.0
 - Permite no tener que instalar servidores web
- Jax-ws (<https://jax-ws.dev.java.net/>)
- Tomcat



Tarea 1 (15 minutos)

- Probar el servicio calculadora proporcionado (directorio FromJava/ejemplo1. Calculadora)
- Crear un cliente para este servicio, que reste, en lugar de sumar (directorio ClienteCalculadora)
- Ejecutarlo
- Observar los mensajes con el TCPSMon
 - REST/SOAP??



Tarea 2 (30 minutos)

- Se proporciona un Servicio web Fibonacci en el directorio ejemplo2-Fibonacci

```
@WebService
public class FibonacciImpl {

    public int calcula(int param) { int result=fibo(param);
                                   return result;    }

    int fibo(int p){
        int result;
        switch(p) {... result=...;}
        return result;
    }
}
```




Tarea 2 (30 minutos)

- Crear un cliente para este servicio
- Ejecutarlo
- Observar los mensajes con el TCPPMon



Tarea 3 (10 minutos) //OPCIONAL TRABAJO DEL ALUMNO

- Probar el ejemplo 3 para utilizar el estilo Document/literal
 - ☐ Necesario utilizar la interfaz JAX-WS's Dispatch para el cliente
 - ☐ Se proporcionar un fichero WSDL para publicitar la carga que deben llevar los mensajes (payload)
 - ☐ El servidor debe implementar la interfaz indicada en el elemento portType del WSDL
- Probarlo (con el mismo cliente u otro) y comparar los mensajes intercambiados con los del servicio anterior.




Tarea 3 (y II) (10 minutos)

- ¿ Qué interés puede tener construir clientes y servicios de esta manera?



Tarea 4 (10 minutos)

- Ejemplo fromWSDL/ejemplo 4
 - ☐ Muestra el modo de programación wsdl→java
 - ☐ Probarlo. ¿qué ficheros hay originalmente? ¿cuáles se generan?
 - ☐ ¿cómo es el cliente? ¿Y la implementación del servicio?
- Tarea extra: Jugar con los ficheros de configuración
 - ☐ cambiando el nombre por defecto “fromwsdl” por “ejemplo4”
 - ☐ Eliminar la operación oneWayInt



Tarea 5 (15 minutos)

- <https://jax-rpc.dev.java.net/docs/wsdl-bindings.html>
- Ejemplo 5
 - Se proporciona un wsdl
 - Ficheros sun-jaxws.xml: se define el endpoint
 - Genere los stubs para el servicio (lea el readme)
 - Codifique el servicio (fíjese en el ejemplo 4)
 - Despliegue el servidor
- Premio para el que descubra por qué en el fichero wsdl la operación y el elemento que describe el mensaje de entrada **SE LLAMAN IGUAL**
- Codifique un cliente
- Pruébalo
- Estudie los mensajes intercambiados



Tarea 6 (15 minutos)

- <https://jax-rpc.dev.java.net/docs/wsdl-bindings.html>
- Ejemplo from WSDL/ejemplo6
 - Se proporciona un wsdl
 - Ficheros sun-jaxws.xml: se define el endpoint
 - Genere los stubs para el servicio (lea el readme)
 - OJO: style="rpc". Mire el WSDL antes de empezar
 - Codifique el servicio (fíjese en el ejemplo 4)
 - Despliegue el servidor
- Codifique un cliente
- Pruébalo
- Estudie los mensajes intercambiados. Compárelos con el ejercicio anterior.
- Determine los posibles casos de uso de cada codificación del mensaje SOAP



Tarea 7 (30 minutos)

- Ejemplo fromWSDL/ejemplo7
 - ☐ Se proporciona un wsdl y los ficheros de configuración necesarios.
- Descripción de la aplicación Calculadora
 - ☐ Suma, resta, multiplica y divide dos números enteros
 - ☐ Simplificaciones
 - Suponga que el divisor nunca es 0 (No lo pruebe¡¡)
 - Suponga que debe retornar el cociente entero
- Complete el fichero wsdl para describir la aplicación Calculadora. Llame:
 - ☐ A cada mensaje Mxxx
 - ☐ A cada operación, Oyyy
 - ☐ A cada portType NombreAplicacionPortType
 - ☐ A cada binding NombreAplicacionBinding
 - ☐ Al servicio NombreAplicacionWS
 - ☐ Codifique el servicio (fíjese en el ejemplo 4)
 - ☐ Despliegue el servidor
- Codifique un cliente
- Pruébelo y estudie los mensajes intercambiados
- Calma, tómese el fin de semana.



Gestión de excepciones

- Una excepción es cualquier evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de instrucciones
- Cuando la excepción ocurre en un servicio web, se propaga al cliente como un SOAP fault.
- Un servicio web puede lanzar una excepción por varias razones:
 - ☐ RuntimeException
 - ☐ RemoteException
 - ☐ SOAPFaultException
 - ☐ User Defined Exception



Excepciones definidas por el usuario

- Es posible que el servicio quiera lanzar una excepción personalizada
- Y el cliente quiera hacer algo más con el mensaje que ha recibido
- Para conseguirlo, podríamos lanzar en el servicio un excepción de usuario con algún formato específico
 - Basándose en la excepción que recibe, el cliente puede decidir qué hacer con el mensaje.
- Las excepciones definidas por el usuario lanzadas dentro del servicio web se describen en el elemento `wsdl:fault` en el documento WSDL del servicio web

<wsdl:fault>

- El elemento fault es uno de los hijos de operation (hermano de input y output): Indica que la operación puede fallar

```
<wsdl:fault name="nmtoken"> *  
  <-- extensibility element (5) --> *  
</wsdl:fault>
```

- Debe tener un único <part>

- Para especificar el formato y contenido que se devuelve en caso de error: <soap:fault>

```
<definitions .... >  
  <binding .... >  
    <operation .... >  
      <fault>*  
        <soap:fault name="nmtoken" use="literal|encoded"  
          encodingStyle="uri-list"? namespace="uri"?>  
      </fault>  
    </operation>  
  </binding>  
</definitions>
```

- El atributo name relaciona este mensaje con el definido en la etiqueta <wsdl:fault> de la operación



Ejemplo

```
...
<element name="AddNumbersFault" type="tns:AddNumbersFault" />
<complexType name="AddNumbersFault">
  <sequence>
    <element name="faultInfo" type="xsd:string" />
    <element name="message" type="xsd:string" />
  </sequence>
</complexType>

<message name="addNumbersFault">
  <!-- un elemento part y sólo uno -->
  <part name="AddNumbersFault" element="tns:AddNumbersFault" />
</message>

<portType name="AddNumbersPortType">
  <operation name="addNumbers">
    <input message="tns:addNumbers" name="add"/>
    <output message="tns:addNumbersResponse" name="addResponse"/>
    <fault name="addNumbersFault" message="tns:addNumbersFault"/>
  </operation>
</portType>
```



Ejemplo

```
<binding name="AddNumbersBinding" type="tns:AddNumbersPortType">
  <soap:binding transport=http://schemas.xmlsoap.org/soap/http
    style="document" />

  <operation name="addNumbers">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
    <fault name="addNumbersFault">
      <soap:fault name="addNumbersFault" use="literal" />
    </fault>
  </operation>
</binding>
```



Tarea 4 (revisited)

- Edite el wsdl
- Fíjese en las etiquetas <fault>
 - ☐ ¿Qué operaciones van a lanzar excepciones?
 - ☐ Codifique el cliente para que lance la excepción y el servicio para que envíe el mensaje que usted quiera.
- Despliegue, pruebe y fíjese en los mensajes intercambiados



Tarea 1 (revisited)

- Sin WSDL también se puede hacer
- Codificamos las clases Exception
- El cliente puede emplearlas en un bloque `try { .. } catch`
- ¿Alguien se imagina cómo llega el cliente a saber los detalles de la Exception personal que hemos codificado?



Tarea 8

- Calculadora con un único método operar.
 - Codifique una excepción UndefinedOperation que se lance en caso de que la operación solicitada no exista. Ponga el mensaje que quiera, pero que indique la causa de la excepción y los parámetros.



Desarrollo de clientes con la API Dispatch

- Pensada para desarrolladores avanzados de XML que prefieren trabajar a bajo nivel con los mensajes (`java.lang.transform.Source` o `javax.xml.soap.SOAPMessage`).
- Un cliente que quiera trabajar con Servicios Web en modo REST, DEBE usar la API Dispatch con enlazado XML/HTTP.
- No se cuenta con ningún documento WSDL o Schema o cualquier otro conocimiento para construir el cliente



Desarrollo de clientes con la API Dispatch

■ ¿ Cómo vamos de tiempo?

☐ BIEN: Leer

http://www.ibm.com/developerworks/websphere/library/techarticles/0707_thaker/0707_thaker.html

☐ Y buscar casos de uso para este modo de desarrollo

☐ MAL: Nos lo saltamos

☐ ARGHH: Haremos lo que podamos



Desarrollo de clientes con la API Dispatch

- Pasos para invocar una operación en un endpoint utilizando la API Dispatch
 - ☐ Crear y configurar una instancia de Service
 - ☐ Crear un cliente Dispatch
 - ☐ Configurar el contexto de la solicitud (request context)
 - ☐ Componer el mensaje
 - ☐ Invocar la operación
 - ☐ Procesar el mensaje de respuesta
- Ejemplo: Servicio HelloWorldService con un único port llamado HelloPort. El portType helloWorldSEI define una única operación hello (document-style, transmite el mensaje hello y recibe el mensaje helloResponse)



Desarrollo de clientes con la API Dispatch

- Paso 1: Crear y configurar una instancia de Service
 - La clase Service es una abstracción que representa un servicio WSDL
 - Service.create(QName serviceName)
 - Crea una instancia dinámica (configuración dinámica, contraposición a la instancia estática que crean las herramientas jax-ws)
 - Retorna un objeto servicio para un servicio con el nombre dado, sin WSDL asociado

```
//Qnames for service
QName serviceName =
    new QName("http://www.example.com/services/HelloWorld",
        "HelloService");

//Create a dynamic Service instance
Service service = Service.create(serviceName);
```

Desarrollo de clientes con la API Dispatch

- Crear y configurar una instancia de Service
 - Un servicio es una colección de ports (interfaz soportada en una dirección bajo un protocolo y reglas de transporte).
 - Añadir un puerto o interfaz al servicio.

```
QName portName =  
    new QName("http://www.example.com/services/HelloWorld",  
        "HelloPort");  
  
//Endpoint Address  
String endpointAddress =  
    "http://localhost:8080/hello/services/HelloService";  
  
// Add a port to the Service  
service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING,  
    endpointAddress);
```



Desarrollo de clientes con la API Dispatch

- Crear y configurar una instancia de Service

```
//Qnames for service
QName serviceName = new
    QName("http://www.example.com/services/HelloWorld", "HelloService");

//Create a dynamic Service instance
Service service = Service.create(serviceName);

QName portName = new QName("http://www.example.com/services/HelloWorld",
    "HelloPort");

//Endpoint Address
String endpointAddress =
    "http://localhost:8080/hello/services/HelloService";

// Add a port to the Service
service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING,
    endpointAddress);
```

Desarrollo de clientes con la API Dispatch

■ Paso 2: Crear un cliente Dispatch

- La interfaz Dispatch da soporte a la invocación dinámica de las operaciones de un endpoint.
- Interfaz javax.xml.ws.Service actúa como factoría para crear instancias Dispatch
- La API Dispatch requiere que el cliente construya los mensajes XML
- Dos modos de uso identificados por las constantes(<http://java.sun.com/javase/6/docs/api/javax/xml/ws/Service.Mode.html>)

■ * javax.xml.ws.Service.Mode.PAYLOAD

- Clientes trabajan con el contenido de <soap:Body> (payload) no con el mensaje completo
- El entorno de tiempo de ejecución JAX-WS es el responsable de proporcionar el sobre SOAP si el servicio utiliza un binding SOAP

//Create a dispatch instance

```
Dispatch<SOAPMessage> dispatch =  
    service.createDispatch(portName, javax.xml.transform.Source.class,  
        Service.Mode.PAYLOAD);
```

■ * javax.xml.ws.Service.Mode.MESSAGE

- Clientes trabajan con estructuras específicas del protocolo que representan a los mensajes
- Es el cliente el responsable de conseguir tanto el sobre como el payload del mensaje

//Create a dispatch instance

```
Dispatch<SOAPMessage> dispatch =  
    service.createDispatch(portName, SOAPMessage.class,  
        Service.Mode.MESSAGE);
```



Desarrollo de clientes con la API Dispatch

- Paso 3: Configurar el contexto de la solicitud (request context)
 - La interface BindingProvider permite a los clientes mantener contextos separados para las fases de petición y respuesta de un mensaje
 - El objeto **request** y el objeto **response** son ambos de tipo Map<String, Object> y se obtienen con los métodos getRequestContext y getResponseContext de la interface BindingProvider

```
//get Binding provider
BindingProvider bp = (BindingProvider) dispatch;

// Configure RequestContext (optional step)
Map<String, Object> rc = bp.getRequestContext();

//Ask RequestContext to USE SOAPAction.
rc.put(BindingProvider.SOAPACTION_USE_PROPERTY, Boolean.TRUE);

//Add SOAPAction to RequestContext
rc.put(BindingProvider.SOAPACTION_URI_PROPERTY, "hello");
```




Desarrollo de clientes con la API Dispatch

- Paso 3: Configurar el contexto de la solicitud (request context)
- Tarea: buscar otras propiedades y sus usos



Desarrollo de clientes con la API Dispatch

■ Paso 4: Componer el mensaje

- Por ejemplo, con la API SOAPMessage (ojo, hay otras)

```
// Obtain a preconfigured SAAJ MessageFactory
MessageFactory factory =
    ((SOAPBinding)
    bp.getBinding()).getMessageFactory();
//Create SOAPMessage Request
SOAPMessage request = factory.createMessage();

//Request Header
SOAPHeader header = request.getSOAPHeader();

//Request Body
SOAPBody body = request.getSOAPBody();
```

Desarrollo de clientes con la API Dispatch

- Componer el mensaje

- Parte del wsdl que representa la signature de la operación

```
<schema xmlns=http://www.w3.org/2001/XMLSchema
  targetNamespace="http://www.example.com/schemas/HelloWorld">
  <element name="hello">
    <complexType> <sequence>
      <element name="message" type="string" />
    </sequence>
  </complexType>
</element>
</schema>
```

```
// Compose the soap:Body payload
QName payloadName =
  new QName("http://www.example.com/schemas/HelloWorld", "hello", "ns1");

SOAPBodyElement payload = body.addBodyElement(payloadName);
SOAPElement message = payload.addChildElement("message");
message.addTextNode("Hello World!");
```



Desarrollo de clientes con la API Dispatch

■ Paso 5: Invocar la operación

- Distintos modelos de patrones de intercambio de mensajes: request-response, request-only,...

```
try { //Invoke Endpoint Operation and read  
response reply = dispatch.invoke(request);  
} catch (WebServiceException wse){  
    wse.printStackTrace(); }
```

- Si la solicitud se despacha con éxito se lanza una `WebServiceException`
- Si el endpoint tiene algún problema procesando la solicitud, se lanza una `SOAPFaultException`

Desarrollo de clientes con la API Dispatch

■ Paso 6: Procesar el mensaje de respuesta

- Extraer el body del mensaje SOAP de respuesta y parsearlo de acuerdo a la definición WSDL
- Respuesta esperada

```
<element name="helloResponse">
```

```
  <complexType>
```

```
    <sequence>
```

```
      <element name="message" type="string" />
```

```
    </sequence>
```

```
  </complexType>
```

```
</element>
```

- Procesamiento

```
// process the reply
```

```
body = reply.getSOAPBody();
```

```
QName responseName =
```

```
    new QName("http://www.example.com/schemas/HelloWorld", "helloResponse");
```

```
SOAPBodyElement bodyElement = body.getChildElements(responseName).next();
```

```
String message = bodyElement.getValue();
```



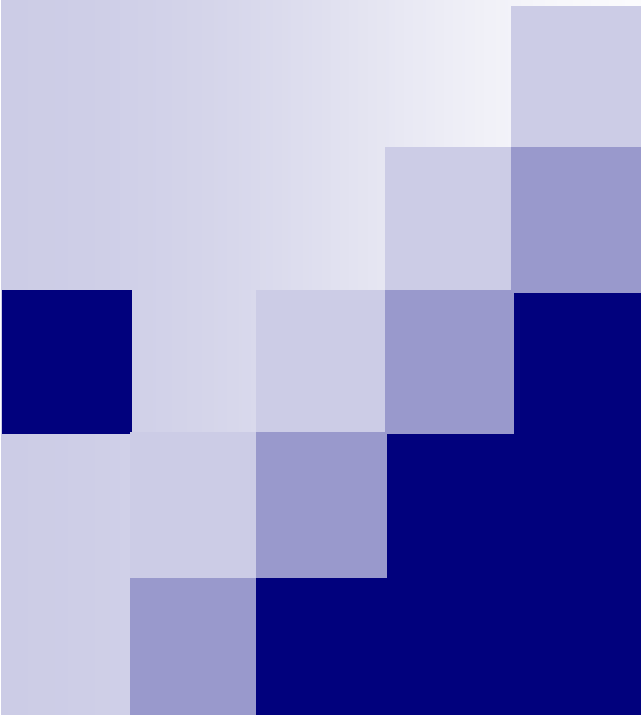
Tarea 3 (10 minutos)

- Cambia los argumentos de la petición.
- Cambia alguna etiqueta de la petición.
 - ☐ Por ejemplo, en lugar de addNumbers, pon suma
- Probarlo (con el mismo cliente u otro) y comparar los mensajes intercambiados



Tarea 9: REST

- Estudiar el ejemplo 9
 - ¿Qué hay diferente con el ejemplo anterior?
- Observar los mensajes con el TCPMon
- Modificar los argumentos
- Añadir una operación al servidor
- Codificar un cliente que invoque esa operación
- ¡¡ Que lance una excepción !!



WSDL (Web Services Description Language) Construcción de servicios web

Lourdes Tajés Martínez
(tajes@uniovi.es)