Introducción a los Servicios Web

Índice

1 ¿Qué es un Servicio Web?....2
2 Características de los Servicios Web....2
3 Arquitecturas Orientadas a Servicios....2
4 Arquitectura de los Servicios Web....4
5 Tecnologías básicas....4
5.1 SOAP....4
5.2 WSDL....6
5.3 UDDI....9
5.4 Tecnologías de segunda generación....10
5.5 WS-I Basic Profile....10
6 Tecnologías J2EE para Servicios Web....11
6.1 JAXP....11
6.2 JAXM....11
6.3 JAX-RPC / JAX-WS....12
6.4 JAXR....12
6.5 JAXB....12
6.6 Otras librerías....13
7 Probar servicios web con Eclipse....14
8 Invocación de Servicios....16
8.1 Tipos de acceso....17
8.2 Invocación mediante stub estático....17
8.3 Interfaz de invocación dinámica (DII)....30

Copyright © 2008-2009 Depto. CCIA All rights reserved.

El diseño del software tiende a ser cada vez más modular. Las aplicaciones se componen de una serie de componentes (servicios) reutilizables, que pueden encontrarse distribuidos a lo largo de una serie de máquinas conectadas en red.

Los Servicios Web nos permitirán distribuir nuestra aplicación a través de Internet, pudiendo una aplicación utilizar los servicios ofrecidos por cualquier servidor conectado a Internet.

1. ¿Qué es un Servicio Web?

Un Servicio Web es un componente al que podemos acceder mediante protocolos Web estándar, utilizando XML para el intercambio de información.

Normalmente nos referimos con Servicio Web a una colección de procedimientos (métodos) a los que podemos llamar desde cualquier lugar de Internet o de nuestra intranet, siendo este mecanismo de invocación totalmente independiente de la plataforma que utilicemos y del lenguaje de programación en el que se haya implementado internamente el servicio.

Cuando conectamos a un servidor web desde nuestro navegador, el servidor nos devuelve la página web solicitada, que es un documento que se mostrará en el navegador para que lo visualice el usuario, pero es difícilmente entendible por una máquina. Podemos ver esto como web para humanos. En contraposición, los Servicios Web ofrecen información con un formato estándar que puede ser entendido fácilmente por una aplicación. En este caso estaríamos ante una web para máquinas.

2. Características de los Servicios Web

Las características deseables de un Servicio Web son:

- Un servicio debe poder ser **accesible a través de la Web**. Para ello debe utilizar protocolos de transporte estándares como HTTP, y codificar los mensajes en un lenguaje estándar que pueda conocer cualquier cliente que quiera utilizar el servicio.
- Un servicio debe contener una **descripción de sí mismo**. De esta forma, una aplicación podrá saber cuál es la función de un determinado Servicio Web, y cuál es su interfaz, de manera que pueda ser utilizado de forma automática por cualquier aplicación, sin la intervención del usuario.
- Debe poder **ser localizado**. Deberemos tener algún mecanismo que nos permita encontrar un Servicio Web que realice una determinada función. De esta forma tendremos la posibilidad de que una aplicación localice el servicio que necesite de forma automática, sin tener que conocerlo previamente el usuario.

3. Arquitecturas Orientadas a Servicios

Copyright © 2008-2009 Depto. CCIA All rights reserved.

Las arquitecturas orientadas a servicios (SOA) se basan en el desarrollo de servicios altamente reutilizables, y en la combinación de estos servicios para dar lugar a nuestra aplicación.

Estos servicios idealmente deberían tener una interfaz estándar bien definida, de forma que se pueda integrar fácilmente en cualquier aplicación. Además no debe tener estado, ni depender del estado de otros componentes. Debe recibir toda la información necesaria en la petición.

Se conoce como *orquestación* de servicios la secuenciación de llamadas a diferentes servicios para realizar un determinado proceso de negocio. Al no tener estado, los servicios se podrán secuenciar en cualquier orden, pudiendo formar así diferentes flujos que implementen la lógica de negocio.

Normalmente cuando hablamos de arquitecturas orientadas a servicios pensamos en su implementación mediante servicios web. Sin embargo, estas arquitecturas puedes estar formadas por cualquier tipo de servicio, como pueden ser por ejemplo servicios accesibles mediante JMS. En el caso de una SOA implementada mediante Servicios Web, sus servicios serán accesibles a través de la web.

En una arquitectura orientada a servicios podemos distinguir tres agentes con diferentes funciones:

Proveedor de servicio	Implementa unas determinadas operaciones (servicio). Un cliente podrá solicitar uno de estos servicios a este proveedor.
Cliente del servicio	Invoca a un proveedor de servicio para la realización de alguna de los operaciones que proporciona.
Registro de servicios	Mantiene una lista de proveedores de servicios disponibles, junto a sus descripciones.

El mecanismo básico de invocación de servicios consistirá en que un cliente solicitará un determinado servicio a un proveedor, efectuando el proveedor dicho servicio. El servidor devolverá una respuesta al cliente como resultado del servicio invocado.

Esto podremos hacerlo así si el cliente conoce de antemano el proveedor del cual va a obtener el servicio. Pero hemos de pensar que en Internet encontraremos una gran cantidad de Servicios Web dispersos, lo cual hará difícil localizar el que busquemos. Además, si hemos localizado uno que realiza la función que necesitamos, si dicho servicio no está mantenido por nosotros puede ocurrir que en algún momento este servicio cambie de lugar, de interfaz o simplemente desaparezca, por lo que no podremos confiar en que vayamos a poder utilizar siempre este mismo servicio.

Los registros de servicios nos permiten automatizar la localización de Servicios Web. Un proveedor puede *anunciarse* en un determinado registro, de forma que figurará en dicho registro la localización de este servicio junto a una descripción de su funcionalidad y de

Copyright © 2008-2009 Depto. CCIA All rights reserved.

su interfaz, que podrá ser entendida por una aplicación.

Cuando un cliente necesite un determinado servicio, puede acudir directamente a un registro y solicitar el tipo de servicio que necesita. Para ello es importante establecer un determinada semántica sobre las posibles descripciones de funcionalidades de servicios, evitando las posibles ambigüedades.

El registro devolverá entonces una lista de servicios que realicen la función deseada, de los cuales el cliente podrá elegir el más apropiado, analizar su interfaz, e invocarlo.

4. Arquitectura de los Servicios Web

Los protocolos utilizados en los Servicios Web se organizan en una serie de capas:

Capa	Descripción
Transporte de servicios	Es la capa que se encarga de transportar los mensajes entre aplicaciones. Normalmente se utiliza el protocolo **HTTP** para este transporte, aunque los servicios web pueden viajar mediante otros protocolos de transferencia de hipertexto como SMTP, FTP o BEEP.
Mensajería XML	Es la capa responsable de codificar los mensajes en XML de forma que puedan ser entendidos por cualquier aplicación. Puede implementar los protocolos XML-RPC o **SOAP**.
Descripción de servicios	Se encarga de definir la interfaz pública de un determinado servicio. Está definición se realiza mediante **WSDL**.
Localización de servicios	Se encarga del registro centralizado de servicios, permitiendo que estos sean anunciados y localizados. Para ello se utiliza el protocolo **UDDI**.

Más adelante describiremos cada una de las tecnologías para Servicios Web vistas en las distintas capas.

5. Tecnologías básicas

Tenemos una serie de tecnologías, todas ellas basadas en XML, que son fundamentales para el desarrollo de Servicios Web. Estas tecnologías son independientes tanto del SO como del lenguaje de programación utilizado para implementar dichos servicios. Por lo tanto, serán utilizadas para cualquier Servicio Web, independientemente de la plataforma sobre la que construyamos dichos servicios (como puede ser J2EE o .NET).

5.1. SOAP

Copyright © 2008-2009 Depto. CCIA All rights reserved.

Se trata de un protocolo derivado de XML que nos sirve para intercambiar información entre aplicaciones.

Normalmente utilizaremos SOAP para conectarnos a un servicio e invocar métodos remotos, aunque puede ser utilizado de forma más genérica para enviar cualquier tipo de contenido. Podemos distinguir dos tipos de mensajes según su contenido:

- **Mensajes orientados al documento**: Contienen cualquier tipo de contenido que queramos enviar entre aplicaciones.
- **Mensajes orientados a RPC**: Este tipo de mensajes servirá para invocar procedimientos de forma remota (*Remote Procedure Calls*). Podemos verlo como un tipo más concreto dentro del tipo anterior, ya que en este caso como contenido del mensaje especificaremos el método que queremos invocar junto a los parámetros que le pasamos, y el servidor nos deberá devolver como respuesta un mensaje SOAP con el resultado de invocar el método.

Puede ser utilizado sobre varios protocolos de transporte, aunque está especialmente diseñado para trabajar sobre HTTP.

Dentro del mensaje SOAP podemos distinguir los siguientes elementos:

Elementos de un mensaje SOAP

- Un sobre (`Envelope`), que describe el mensaje, a quien va dirigido, y cómo debe ser procesado. El sobre incluye las definiciones de tipos que se usarán en el documento. Contiene una cabecera de forma opcional, y el cuerpo del mensaje.
- Una cabecera (`Header`) opcional, donde podemos incluir información sobre el mensaje. Por ejemplo, podemos especificar si el mensaje es obligatorio (debe ser entendido de forma obligatoria por el destinatario), e indicar los actores (lugares por donde ha pasado el mensaje).
- El cuerpo del mensaje (`Body`), que contiene el mensaje en si. En el caso de los mensajes RPC se define una convención sobre como debe ser este contenido, en el que se especificará el método al que se invoca y los valores que se pasan como parámetros. Puede contener un error de forma opcional.
- Un error (`Fault`) en el cuerpo del mensaje de forma opcional. Nos servirá para indicar en una respuesta SOAP que ha habido un error en el procesamiento del mensaje de petición que mandamos.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

Hemos visto como los mensajes SOAP nos sirven para intercambiar cualquier documento XML entre aplicaciones. Pero puede ocurrir que necesitemos enviar en el mensaje datos que no son XML, como puede ser una imagen. En ese caso tendremos que recurrir a la especificación de mensajes SOAP con anexos.

Los mensajes SOAP con anexos añaden un elemento más al mensaje:

Elementos de un mensaje SOAP con Anexos

- El anexo (`Attachment`), puede contener cualquier tipo de contenido (incluido el XML). De esta forma podremos enviar cualquier tipo de contenido junto a un mensaje SOAP.

Nuestro mensaje podrá contener tantos anexos como queramos.

Un ejemplo de mensaje SOAP es el siguiente:

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

En él estamos llamando a nuestro método `getTemperatura` para obtener información meteorológica, proporcionando como parámetro el área de la que queremos obtener la temperatura.

Podemos encontrar la especificación de SOAP y SOAP con anexos publicada en la página del W3C, en las direcciones `http://www.w3.org/TR/SOAP/` y `http://www.w3.org/TR/SOAP-attachments` respectivamente.

5.2. WSDL

Copyright © 2008-2009 Depto. CCIA All rights reserved.

Es otro lenguaje derivado de XML, que se utiliza para describir los Servicios Web, de forma que una aplicación pueda conocer de forma automática la función de un Servicio Web, así como la forma de uso de dicho Servicio Web.

El fichero WSDL describirá la interfaz del Servicio Web, con los métodos a los que podemos invocar, los parámetros que debemos proporcionarles y los tipos de datos de dichos parámetros.

Si desarrollamos un Servicio Web, y queremos que otras personas sean capaces de utilizar nuestro servicio para sus aplicaciones, podremos proporcionar un documento WSDL describiendo nuestro servicio. De esta forma, a partir de este documento otros usuarios podrán generar aplicaciones clientes en cualquier plataforma (ya que WSDL se define como un estándar) que se ajusten a nuestro servicio.

El elemento raíz dentro de este fichero es `definitions`, donde se especifican los espacios de nombres que utilizamos en nuestro servicio. Dentro de este elemento raíz encontramos los siguientes elementos:

- `types`: Se utiliza para definir los tipos de datos que se intercambiarán en el mensaje.
- `message`: Define los distintos mensajes que se intercambiaran durante el proceso de invocación del servicio. Se deberán definir los mensajes de entrada y salida para cada operación que ofrezca el servicio. En el caso de mensajes RPC, en el mensaje de entrada se definirán los tipos de parámetros que se proporcionan, y en el de salida el tipo del valor devuelto.
- `portType`: Define las operaciones que ofrece el servicio. De cada operación indica cuales son los mensajes de entrada y salida, de entre los mensajes definidos en el apartado anterior.
- `binding`: Indica el protocolo y el formato de los datos para cada mensaje de los definidos anteriormente. Este formato puede ser orientado al documento u orientado a RPC. Si es orientado al documento tanto el mensaje de entrada como el de salida contendrán un documento XML. Si es orientado a RPC el mensaje de entrada contendrá el método invocado y sus parámetros, y el de salida el resultado de invocar dicho método, siguiendo una estructura más restrictiva.
- `service`: Define el servicio como una colección de puertos a los que se puede acceder. Un puerto es la dirección (URL) donde el servicio actúa. Esta será la dirección a la que las aplicaciones deberán conectarse para acceder al servicio. Además contiene la documentación en lenguaje natural del servicio.

Un documento WSDL de ejemplo es el siguiente:

```
<?xml version="1.0" encoding="utf-8" ?>
  <definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
     xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:tns="http://j2ee.ua.es/wsdl"
     xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
     targetNamespace="http://j2ee.ua.es/wsdl"
```


Copyright © 2008-2009 Depto. CCIA All rights reserved.

```
      xmlns="http://schemas.xmlsoap.org/wsdl/">
    <message name="getTempRequest">
      <part name="string_1"
            xmlns:partns="http://www.w3.org/2001/XMLSchema"
            type="partns:string" />
    </message>
    <message name="getTempResponse">
      <part name="double_1"
            xmlns:partns="http://www.w3.org/2001/XMLSchema"
            type="partns:double" />
    </message>
    <portType name="TempPortType">
      <operation name="getTemp">
        <input message="tns:getTempRequest" />
        <output message="tns:getTempResponse" />
      </operation>
    </portType>
    <binding name="TempPortSoapBinding" type="tns:TempPortType">
      <soap:binding style="rpc"
          transport="http://schemas.xmlsoap.org/soap/http" />
      <operation name="getTemp">
        <soap:operation soapAction=" style="rpc" />
        <input>
          <soap:body use="encoded"
          namespace="http://j2ee.ua.es/wsdl"
          encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/" />
        </input>
        <output>
          <soap:body use="encoded"
          namespace="http://j2ee.ua.es/wsdl"
          encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/" />
        </output>
      </operation>
    </binding>
    <service name="Temp">
      <documentation>Documentacion</documentation>
      <port name="TempPort" binding="tns:TempPortSoapBinding">
        <soap:address
            location="http://localhost:7001/sw_temp/Temp" />
      </port>
    </service>
</definitions>
```

En el que se define un servicio que proporciona el método `getTemp`, que toma como parámetro una cadena con el nombre del área que queremos consultar, y nos devuelve un valor real.

En los elementos `message` vemos que tenemos dos mensajes: los mensajes de entrada y salida de la operación `getTemp` de nuestro servicio. El mensaje de entrada contiene un dato de tipo `string` (el parámetro del método), y el de salida es de tipo `double` (la temperatura que devuelve el servicio).

El elemento `portType` define la operación `getTemp` a partir de los mensajes de entrada y salida que la componen, y en `binding` se establece esta operación como de estilo `rpc`,

Copyright © 2008-2009 Depto. CCIA All rights reserved.

con codificación `encoded`. Esto es lo que se conoce como tipo `rpc/encoded`. Actualmente dicho tipo de servicios está desaprobado. Los servicios web deben ser en su lugar de tipo `document/literal`.

Por último en el apartado `service` se especifica el puerto al que podemos conectar para usar el servicio, dando la URL a la que nuestro cliente deberá acceder.

Podemos encontrar la especificación de WSDL publicada en la página del W3C, en la dirección `http://www.w3.org/TR/wsdl`.

5.3. UDDI

UDDI nos permite localizar Servicios Web. Para ello define la especificación para construir un directorio distribuido de Servicios Web, donde los datos se almacenan en XML. En este registro no sólo se almacena información sobre servicios, sino también sobre las organizaciones que los proporcionan, la categoría en la que se encuentran, y sus instrucciones de uso (normalmente WSDL). Tenemos por lo tanto 3 tipos de información relacionados entre sí:

- *Páginas blancas*: Datos de las organizaciones (dirección, información de contacto, etc).
- *Páginas amarillas*: Clasificación de las organizaciones (según tipo de industria, zona geográfica, etc).
- *Páginas verdes*: Información técnica sobre los servicios que se ofrecen. Aquí se dan las instrucciones para utilizar los servicios. Es recomendable que estas instrucciones se especifiquen de forma estándar mediante un documento WSDL.

Además, UDDI define una API para trabajar con dicho registro, que nos permitirá buscar datos almacenados en él, y publicar datos nuevos.

De esta forma, una aplicación podrá anunciar sus servicios en un registro UDDI, o bien localizar servicios que necesitemos mediante este registro.

Esta capacidad de localizar servicios en tiempo de ejecución, y de que una aplicación pueda saber cómo utilizarlo inmediatamente gracias a la descripción del servicio, nos permitirá realizar una integración débilmente acoplada de nuestra aplicación.

La interfaz de UDDI está basada en SOAP. Para acceder al registro se utilizarán mensajes SOAP, que son transportados mediante protocolo HTTP.

Podemos encontrar la especificación de UDDI, documentación, y más información en la dirección `http://www.uddi.org/`.

Estos registros se utilizan normalmente de forma interna en organizaciones para tener un directorio organizado de servicios. Podemos encontrar varios registros proporcionados por diferentes proveedores. Destacamos **jUDDI**, un registro *open-source* de Apache. Este registro consiste en una aplicación web Java que puede instalarse en cualquier servidor

Copyright © 2008-2009 Depto. CCIA All rights reserved.

con soporte para este tipo de aplicaciones, como puede ser Tomcat, y una base de datos, que podrá ser instalada en diferentes SGBD (MySQL, Postgres, Oracle, etc).

5.4. Tecnologías de segunda generación

Una vez asentadas las tecnologías básicas para servicios web que hemos visto en los puntos anteriores, se empiezan a desarrollar extensiones sobre ellas para cubrir las necesidades que van apareciendo, entre las que encontramos:

- *WS-Policy* y *WS-PolicyAttachment* nos permitirán describir funcionalidades que no podíamos especificar con WSDL.
- *WS-Security* nos permitirá añadir características de seguridad adaptadas a las necesidades de seguridad de los Servicios Web. Con esta API podemos utilizar seguridad a nivel de mensaje (encriptando sólo determinadas partes del mensaje SOAP), mientras que con SSL sólo podríamos hacer que fuese seguro a nivel de transporte.
- *WS-Addressing* y *WS-ReliableMessaging* nos permitirán especificar la dirección de un servicio y realizar un control de flujo de los mensajes respectivamente. Gracias a esto se podrá por ejemplo implementar servicios con estado, o servicios que funcionen de forma asíncrona. Podremos hacer una petición sin quedarnos bloqueados esperando una respuesta, y recibir la respuesta mediante un *callback*.
- *WS-Coordination* o *BPEL* nos permitirán orquestar servicios web.

5.5. WS-I Basic Profile

La *Web Services Interoperability Organization* (WS-I) vela por el cumplimiento de los estándares en los servicios web, de forma que se garantice la interoperabilidad entre servicios web desarrollados en diferentes plataformas.

Hemos visto hasta el momento diferentes estándares que dan soporte a los servicios web (SOAP, WSDL, UDDI). El problema es que dichos estándares contienen algunas ambigüedades, que pueden ser interpretadas de forma diferente por los desarrolladores de diferentes plataformas, y que tampoco se especifica como usar las tres tecnologías de forma conjunta para el desarrollo de servicios web (sólo se dan las especificaciones por separado). Para solucionar este problema la WS-I crea la especificación WS-I Basic Profile, para aclarar la forma en la que estas tecnologías deben ser utilizadas cuando las utilizamos para construir servicios web.

En el WS-I Basic Profile también se especifica como deben combinarse las tecnologías SOAP, WSDL y UDDI para el desarrollo de servicios web. Utilizaremos un registro UDDI para publicar los servicios. De estos servicios se publicará un documento WSDL como instrucciones de uso. En el WSDL figurará la interfaz del servicio, y la forma de acceder a él, que será mediante protocolo SOAP.

Por lo tanto, cuando desarrollemos servicios web que cumplan con dicho perfil, podremos

Copyright © 2008-2009 Depto. CCIA All rights reserved.

tener la certeza de que podrán ser accedidos por clientes que también lo cumplan.

6. Tecnologías J2EE para Servicios Web

Hemos visto las tecnologías en las que se basan los Servicios Web, y que los hacen independientes de la plataforma y del lenguaje de programación utilizado. Sin embargo, escribir manualmente los mensajes SOAP desde nuestras aplicaciones puede ser una tarea tediosa. Por ello, las distintas plataformas existentes incorporan librerías y utilidades que se encargan de realizar esta tarea por nosotros.

En este tema veremos las librerías que incorpora Java EE para la generación y el procesamiento de código XML, que nos servirán para implementar y utilizar Servicios Web.

Hemos de destacar que las tecnologías Java para servicios web cumplen con el WS-I Basic Profile, por lo que tenemos garantizada la interoperabilidad con gran parte de los servicios web desarrollados en otras plataformas.

6.1. JAXP

La API JAXP nos permite procesar cualquier documento XML desde lenguaje Java. Tiene en cuenta los espacios de nombres, lo cual nos permite trabajar con DTDs que podrían tener conflictos de nombres si estos no estuviesen soportados. Además, soporta XSLT, lo cual nos permitirá convertir un documento XML a otro formato, como por ejemplo HTML.

Esta es una librería genérica, para procesar cualquier documento XML. A continuación veremos una serie de librerías, para tareas más especificas, que se apoyan en JAXP para realizar el procesado de diferentes lenguajes como SOAP, WSDL y UDDI, todos ellos derivados de XML. Por lo tanto, todas estas librerías dependerán de JAXP para su correcto funcionamiento.

6.2. JAXM

La API JAXM implementa la mensajería XML en Java orientada al documento. Nos permitirá de forma sencilla crear mensajes XML, insertando el contenido que queramos en ellos, y enviarlos a cualquier destinatario, así como extraer el contenido de los mensajes que recibamos. Permite enviar y recibir los mensajes de forma síncrona (modelo petición-respuesta) o asíncrona (envío de mensaje sin esperar respuesta).

Los mensajes XML con los que trabaja JAXM siguen la especificación SOAP y SOAP con anexos. Dentro de JAXM encontramos dos APIs:

- SAAJ (*SOAP with Attachmets API for Java*) es la API que se utiliza para construir mensajes SOAP y para extraer la información que contienen. Esta API es

Copyright © 2008-2009 Depto. CCIA All rights reserved.

independiente, y suficiente para enviar mensajes de tipo petición-respuesta (síncronos).

- JAXM proporciona un proveedor de mensajería XML, con el que podremos enviar y recibir mensajes de forma asíncrona, sin necesidad de esperar una respuesta de la otra parte. Esta API dependerá de SAAJ para funcionar, ya que SAAJ es la que se encargará de crear y manipular los mensajes.

6.3. JAX-RPC / JAX-WS

La API JAX-RPC implementa la infraestructura para realizar llamadas a procedimiento remoto (RPC) mediante XML. En este caso se enviará un mensaje SOAP con el método que queremos invocar junto a los parámetros que le pasamos, y nos devolverá de forma síncrona una respuesta SOAP con el valor devuelto por el método tras su ejecución.

Por lo tanto, JAX-RPC dependerá de SAAJ para construir los mensajes SOAP, para enviarlos, y para extraer la información del mensaje SOAP que nos devuelve como resultado.

Esta API nos permitirá, de forma sencilla, invocar Servicios Web, así como crear nuestros propios Servicios Web a partir de clases Java que tengamos implementadas.

A partir de la versión 2.0, esta API pasa a recibir el nombre JAX-WS. Esta nueva versión se basa en JAXB para manipular los datos. Además permite el uso de la API *Web Services Metadata for the Java Platform* que permite construir los servicios web utilizando anotaciones.

6.4. JAXR

La API JAXR nos permitirá acceder a registros XML a través de una API estándar Java. Esta API pretende proporcionar una interfaz única para acceder a distintos tipos de registros, cada uno de los cuales tiene un protocolo distinto.

Actualmente JAXR es capaz de trabajar con registros UDDI y ebXML. Podremos realizar dos tipos de tareas distintas cuando accedamos a un registro mediante JAXR:

- Consultar el registro, para localizar los servicios que necesitemos.
- Publicar un servicio en el registro, para que otros clientes sean capaces de localizarlo cuando lo necesiten, así como modificar o eliminar los servicios publicados que sean de nuestra propiedad.

6.5. JAXB

La API de JAXB (*Java API for Binding*) nos permite asociar esquemas XML y código Java. A partir de un esquema XML, podremos generar una clase Java que represente dicho esquema.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

De esta forma podremos convertir un documento XML a una serie de objetos Java que contendrán la información de dicho documento (*unmarshalling*). Podremos entonces trabajar desde nuestra aplicación con estos objetos, accediendo y modificando sus valores. Finalmente, podremos volver a obtener un documento XML a partir de los objetos Java (*marshalling*).

Esto nos va a simplificar la tarea de utilizar tipos de datos propios en llamadas a Servicios Web, ya que utilizando JAXB podremos realizar de forma sencilla la conversión entre nuestra clase Java y un documento XML con la información de dicha clase.

6.6. Otras librerías

Además de las APIs Java estándar para servicios web, encontramos también algunas librerías adicionales desarrolladas por terceros que pueden sernos de utilidad. Por ejemplo, vamos a ver una serie de librerías para trabajar con servicios web que se encuentran enmarcadas dentro del proyecto Apache. Más adelante estudiaremos el framework Axis, también de Apache, que es uno de los frameworks más utilizados para el desarrollo de servicios web en Java.

6.6.1. WSDL4J

La API de Java para WSDL (WSDL4J) nos permite de forma sencilla analizar documentos WSDL, y de esa forma poder descubrir las características de un servicio en tiempo de ejecución.

Mediante esta API, podremos *"interrogar"* un servicio a partir de su especificación WSDL, y obtener información como las operaciones que podemos invocar en este servicio, los parámetros que deberemos proporcionar a cada una de ellas, y el tipo de datos resultante que nos devuelven.

De esta forma podremos realizar la integración de la aplicación en tiempo de ejecución, ya que no será necesario indicar al programa cómo debe acceder a un servicio, ni los métodos a los que debe llamar, sino que el programa será capaz de determinar esta información analizando la especificación WSDL del servicio. Si estamos interesados en acceder a los servicios web de esta forma, lo más inmediato será utilizar directamente la siguiente API.

6.6.2. WSIF

La API *Web Services Invocation Framework* (WSIF) nos va a permitir acceder a los servicios web directamente a partir del WSDL, sin tener que programar el acceso mediante SOAP, ni crear ningún *stub* que haga esto. Gracias a esta API podremos hacer de forma sencilla una integración dinámica y muy débilmente acoplada de cualquier servicio web en nuestras aplicaciones.

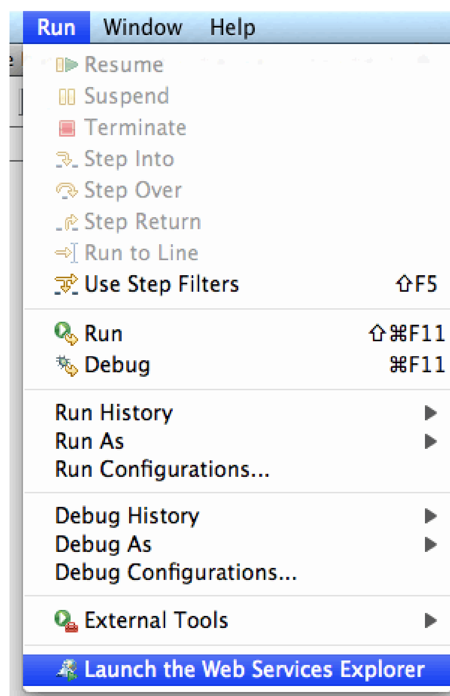
Copyright © 2008-2009 Depto. CCIA All rights reserved.

6.6.3. UDDI4J

Otra de las librerías para servicios web de Apache es UDDI4J. Esta librería nos permite acceder a registros UDDI desde Java. A diferencia de JAXR, esta librería se centra únicamente en UDDI, sin pretender ser una librería genérica para acceso a registros XML. Esto hace que resulte más sencillo trabajar con ella, ya que está adaptada a la nomenclatura utilizada en UDDI.

7. Probar servicios web con Eclipse

Existen numerosas aplicaciones que nos permiten probar servicios web simplemente introduciendo la dirección donde se encuentra el documento WSDL que los define. Una de ellas es *Web Services Explorer*, integrada en el entorno Eclipse. Podemos acceder a ella mediante el menú *Run* > *Web Services Explorer*:

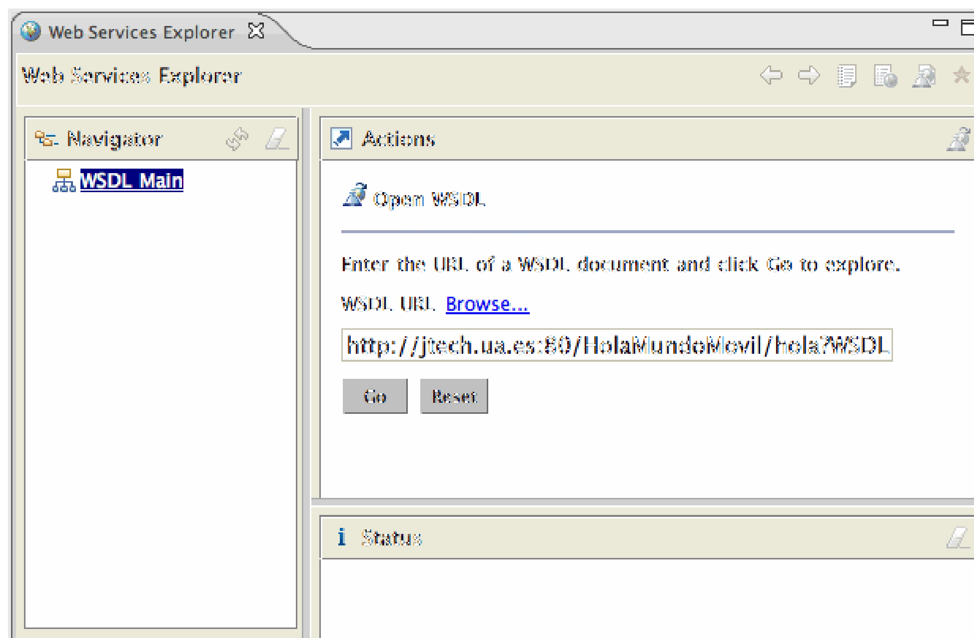


Abrir Web Services Explorer

Una vez abierta esta herramienta, en la barra superior podremos seleccionar diferentes modos: UDDI, WSIL y WSDL. El primero de ellos nos permitirá conectarnos a registros UDDI y explorar su contenido. Con el segundo podremos explorar documentos WSIL (*Web Services Inspection Languaje*). Estos documentos se utilizan para especificar la lista de servicios ofrecidos por un determinado sitio. El tercero de ellos es el que nos permitirá acceder a un servicio proporcionando su correspondiente documento WSDL.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

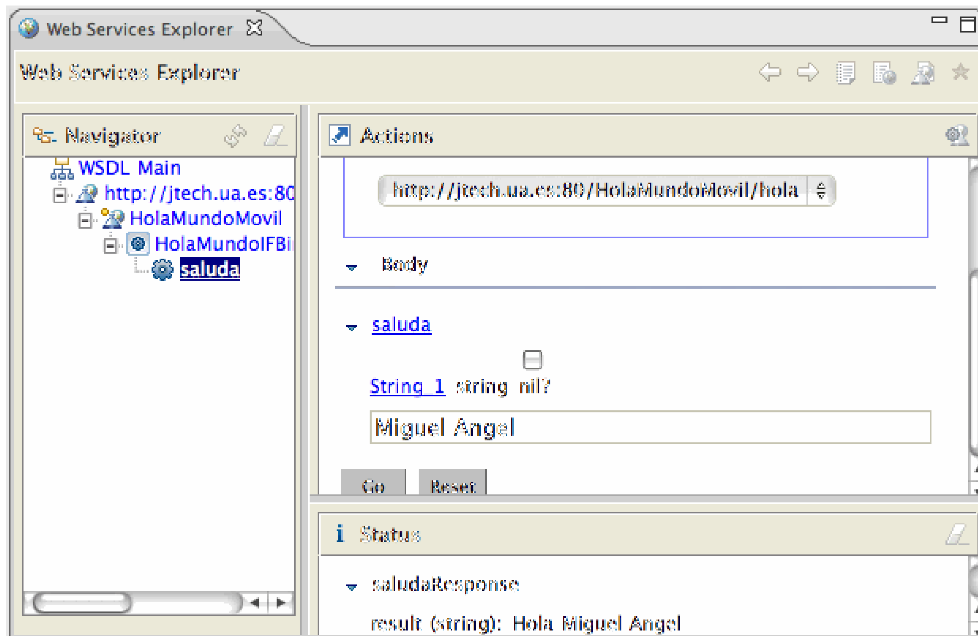
Vamos a entrar en el modo WSDL. Para poder acceder a un servicio, primero pulsaremos sobre *WSDL Main*, y a continuación introduciremos el documento WSDL que lo define.



Seleccionar un documento WSDL

Pulsamos sobre el botón *Go* y veremos que en la sección *Navigator* nos aparece un árbol con los elementos del servicio al que estamos accediento. Podemos desplegarlo para ver las operaciones que nos ofrece, y pulsar sobre una de ellas para invocarla. Al hacer esto en la parte derecha nos aparecerá un formulario con los parámetros de entrada que toma la función que vayamos a ejecutar. Introduciremos un valor para estos parámetros, y pulsaremos sobre el botón *Go* para invocar la operación.

Copyright © 2008-2009 Depto. CCIA All rights reserved.



Ejecutar una operación

En la parte inferior veremos el resultado que nos ha devuelto el servicio. De esta forma vemos como los servicios pueden ser integrados de forma dinámica en tiempo de ejecución.

Esta herramienta nos permitirá probar de forma rápida cualquier servicio, propio o ajeno, sin tener que construir un cliente para él.

8. Invocación de Servicios

Vamos a ver ahora cómo invocar Servicios Web desde Java. Para ello contamos con la API JAX-WS (o JAX-RPC en versiones anteriores). Con esta librería podremos ejecutar procedimientos de forma remota, simplemente haciendo una llamada a dicho procedimiento, sin tener que introducir apenas código adicional. Será JAX-WS quien se encargue de gestionar internamente la conexión con el servicio y el manejo de los mensajes SOAP de llamada al procedimiento y de respuesta.

Podemos encontrar las clases de la API de JAX-WS dentro del paquete `javax.xml.ws` y en subpaquetes de éste.

Nuestro cliente Java realizado con JAX-WS será interoperable con prácticamente todos los servicios creados desde otras plataformas que cumplan el estándar WS-I.

> **Nota:**
> Con JAX-WS no podremos invocar servicios que utilicen codificación `rpc/encoded`, ya que el uso de esta codificación está desaconsejado por los estándares actuales. En estos casos

Copyright © 2008-2009 Depto. CCIA All rights reserved.

deberemos utilizar la antigua librería JAX-RPC, que si que es compatible con este tipo de servicios.

8.1. Tipos de acceso

Tenemos dos formas diferentes de invocar un Servicio Web utilizando JAX-WS o JAX-RPC:

- **Creación de un stub estático**: Consiste en generar una capa de *stub* por debajo del cliente de forma automática. Dicho *stub* implementará la misma interfaz que el servicio, lo cuál nos permitirá desde nuestro cliente acceder al Servicio Web a través del *stub* tal y como si estuviéramos accediendo directamente al servicio.

  Para utilizar este mecanismo es recomendable contar con alguna herramienta dentro de nuestra plataforma que nos permita generar dicho *stub*, para no tener que encargarnos nosotros de realizar esta tarea manualmente.
- **Utilización de la Interfaz de Invocación Dinámica (DII)**: Esta forma de acceso nos permitirá hacer llamadas a procedimientos de nuestro Servicio Web de forma dinámica, sin crear un *stub* para ello. Utilizaremos este tipo de invocación cuando no conozcamos la interfaz del servicio *a priori*, para invocarlo deberemos proporcionar únicamente los nombres de los métodos a utilizar mediante una cadena de texto.

  Podremos utilizar esta interfaz dinámica aunque no contemos con un documento WSDL que nos indique la interfaz y datos de nuestro servicio. En este caso, deberemos proporcionar manualmente esta información, de forma que sea capaz de acceder al servicio correctamente.

8.2. Invocación mediante stub estático

Está será la forma más sencilla de acceder siempre que contemos con una herramienta que genera el *stub* de forma automática.

De esta forma, una vez generado el *stub*, sólo tendremos que utilizar este *stub* como si se tratase de nuestro servicio directamente. En el *stub* podremos hacer las mismas llamadas a métodos que haríamos directamente en la clase que implemente nuestro servicio, ya que ambos implementarán la misma interfaz.

Las herramientas para generar este *stub* variarán según la plataforma con la que trabajemos. A partir de JDK 1.6 se incluye en Java SE la librería JAX-WS y las herramientas necesarias para crear e invocar servicios. En el caso de contar con versiones anteriores de JDK, podemos o bien incorporar a nuestro proyecto las librerías necesarias (JAX-WS o JAX-RPC), o bien utilizar herramientas y librerías similares proporcionadas por los servidores de aplicaciones, como es el caso de Weblogic 9.2, o desarrolladas por terceros, como es el caso de Apache Axis, que se incluye en Eclipse Web Tools.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

Vamos a ver cómo crear clientes para servicios web utilizando tanto las herramientas estándar proporcionadas por Sun (en JDK 1.6), como las herramientas proporcionadas por Weblogic. Finalmente, veremos como crear estos clientes utilizando el IDE Netbeans, que internamente utilizará las herramientas y librerías estándar de Sun.

8.2.1. Invocación de servicios web con JDK 1.6

Para crear un cliente en JDK 1.6 (o con JAX-WS en versiones anteriores de JDK) utilizaremos la herramienta `wsimport`, que tomará como entrada el documento WSDL del servicio al que queremos acceder y producirá un conjunto de clases Java que nos permitirán acceder al servicio. Esta herramienta se puede invocar desde línea de comando:

```
wsimport -s <src.dir> -d <dest.dir> -p <pkg> <wsdl.uri>
```

El documento WSDL (`<wsdl.url>`) se especificará mediante su ruta en el disco o mediante su URL. Podemos proporcionar otros parámetros para indicar la forma en la que se debe generar el *stub*, como el directorio donde queremos que guarde los fuentes de las clases generadas (`<src.dir>`), el directorio donde guardará estas clases compiladas (`<dest.dir>`), y el paquete en el que se generará este conjunto de clases (`<pkg>`).

Por ejemplo podríamos utilizar el siguiente comando para crear el cliente de un servicio `HolaMundo` que se encuentra definido en `http://jtech.ua.es/HolaMundo/wsdl/HolaMundoSW.wsdl`, separando los fuentes en el directorio `src` y las clases compiladas en `bin`, y generando todas estas clases dentro de un paquete `es.ua.jtech.servcweb.hola.stub`:

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

También existe una tarea de ant equivalente. Para utilizarla deberemos declararla previamente (teniendo la librería `jaxws-tools.jar` dentro del `CLASSPATH` de ant):

```
<taskdef name="wsimport"
classname="com.sun.tools.ws.ant.WsImport"/>
```

Una vez declarada, podremos utilizarla de forma similar a la herramienta en línea de comando:

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

Con esto se generarán una serie de clases que nos permitirán acceder al servicio web e invocar sus operaciones desde nuestro cliente. Dentro de estas clases tendremos una que recibirá el mismo nombre que el servicio, y que heredará de la clase `Service`. Deberemos instanciar esta clase, y a partir de ella obtendremos el *stub* para acceder a un puerto del servicio. Este *stub* tendrá la misma interfaz que el servicio web, y a partir de él podremos invocar sus operaciones. En el ejemplo del servicio `HolaMundo` accederíamos al servicio

Copyright © 2008-2009 Depto. CCIA All rights reserved.

de la siguiente forma:

```
HolaMundoSWService service = new HolaMundoSWService();
HolaMundoSW port = service.getHolaMundoSW();

System.out.println("Resultado: " + port.saluda("Miguel"));
```

El servicio ofrece una operación `saluda` que recibe un nombre, y genera un saludo incluyendo dicho nombre. En este ejemplo vemos como podemos invocar la operación `saluda` del servicio a partir del *stub* (`port`) como si se tratase de un método de un objeto local.

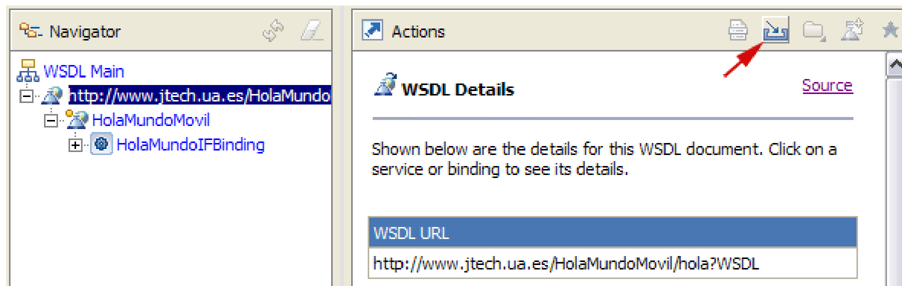
8.2.2. Invocación de servicios web con Eclipse (Axis)

Una forma más cómoda de acceder a servicios web es utilizar los asistentes que nos proporcionan los diferentes IDEs para generar los clientes de los servicios de forma automática. En Eclipse estas facilidades se implementan mediante el framework Axis de Apache, en lugar de utilizar las librerías de Sun. Este framework Java de servicios web se encuentra en un estado de madurez, y goza actualmente de una gran difusión. Además resulta altamente portable, ya que los servicios web Axis se implementan en una aplicación Java EE estándar que incluye las librerías de este framework, por lo que podrá ser llevada a cualquier servidor con soporte para aplicaciones web Java, como puede ser Tomcat.

Al crear un cliente para servicios web mediante Eclipse veremos que introduce en el proyecto las librerías de Axis, ya que el *stub* se genera utilizando este framework. Vamos a ver paso a paso como crear este cliente.

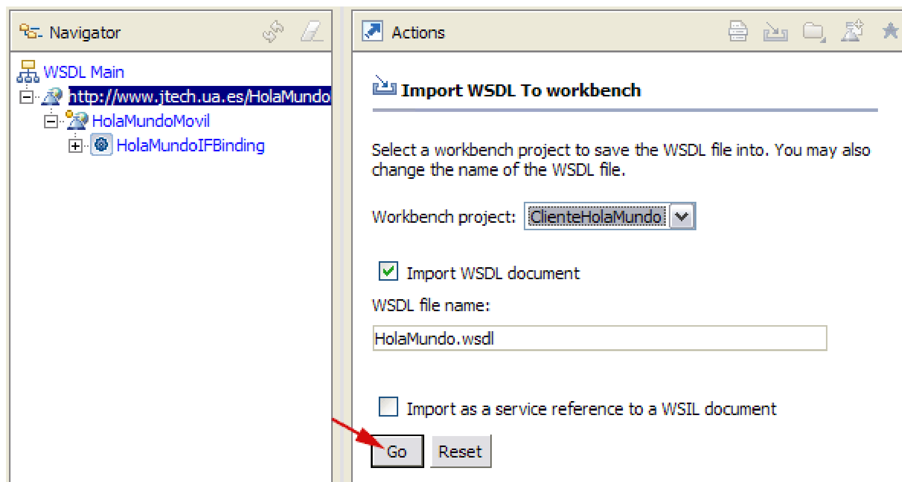
- En primer lugar, si todavía no tenemos ningún proyecto creado, deberemos crear el proyecto en el que introducir el cliente. Podremos utilizar cualquier tipo de proyecto (proyecto Java, proyecto web dinámico, etc), ya que todos ellos van a poder acceder a servicios web.
- Ahora deberemos importar en nuestro proyecto el documento WSDL que define el servicio al que queremos acceder. Podemos hacer esto simplemente arrastrando o copiando este fichero sobre nuestro proyecto. También podremos hacer esto utilizando la herramienta *Web Services Explorer*. Si optamos por esta última forma, abriremos dicha herramienta y accederemos desde ella la URL en la que se encuentra publicado el documento WSDL del servicio que queremos utilizar. Una vez hayamos abierto dicho documento WSDL, pulsaremos sobre el botón para importarlo en nuestro *workspace*.

Copyright © 2008-2009 Depto. CCIA All rights reserved.



Creación de un cliente desde Eclipse. Paso 1.

- Deberemos seleccionar el proyecto de nuestro *workspace* al que queramos importarlo, y pulsar sobre el botón *Go*.



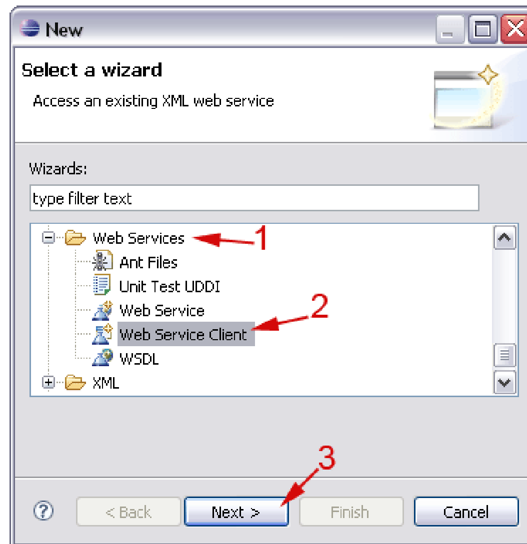
Creación de un cliente desde Eclipse. Paso 2.

- Una vez hecho esto, tendremos el documento WSDL importado en nuestro proyecto.

Creación de un cliente desde Eclipse. Paso 3.

- Ahora es el momento de crear el *stub* para acceder al servicio. Para ello pulsamos con el botón derecho sobre nuestro proyecto y seleccionamos *New > Other* .... En la lista que aparecerá, entramos en la sección *Web Services* y dentro de ella seleccionamos *Web Service Client*. Tras esto, pulsamos sobre *Next* para pasar a la siguiente pantalla del asistente.

Copyright © 2008-2009 Depto. CCIA All rights reserved.



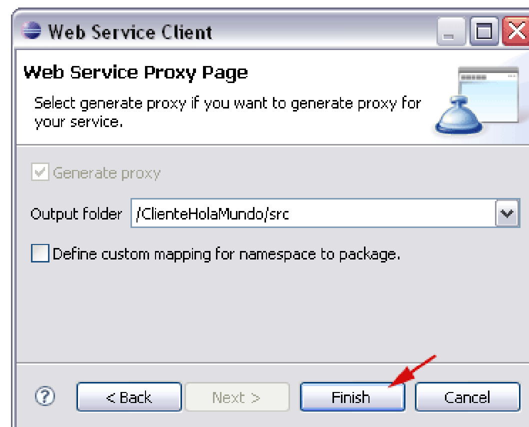
Creación de un cliente desde Eclipse. Paso 4.

- Ahora deberemos especificar el documento WSDL del servicio al que queremos acceder. Pulsamos el botón *Browse...* y seleccionamos el documento que hemos importado en nuestro proyecto. También podremos especificar hasta qué punto queremos que construya el cliente. Será suficiente con seleccionar el nivel mínimo (*Develop client*), ya que con esto creará las clases del *stub* que nosotros posteriormente podremos ejecutar. Tras esto podemos pulsar el botón *Next*.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

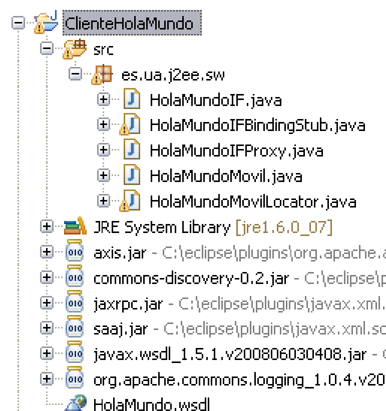
Creación de un cliente desde Eclipse. Paso 5.

- En la siguiente pantalla se especificará el directorio de fuentes en el que queramos que genere el *stub*, y también nos permite especificar de forma manual el paquete al que se mapearán estas clases. Una vez hayamos introducido esta información, pulsaremos sobre *Finish*.



Creación de un cliente desde Eclipse. Paso 6.

- Con esto nos habrá generado las clases del *stub*, y habrá introducido en el proyecto las librerías de Axis necesarias.



Creación de un cliente desde Eclipse. Paso 7.

- Entre las clases del *stub*, veremos una clase con sufijo `BindingStub`, que es en la que realmente se encuentra el código que hace la llamada al servicio. Sin embargo, no accederemos a ella directamente, sino que la referenciaremos mediante la interfaz que nos da acceso al servicio. Tampoco la instanciaremos directamente, sino mediante el localizador, que es la clase con sufijo `Locator`. Por ejemplo, el código para acceder al servicio podría ser el siguiente:

```
HolaMundoMovilLocator locator = new HolaMundoMovilLocator();
HolaMundoIF service = locator.getHolaMundoIFPort();

System.out.println("Resultado: " + service.saluda("Miguel"));
```


Copyright © 2008-2009 Depto. CCIA All rights reserved.

De forma alternativa, podríamos utilizar la clase con sufijo `Proxy` para instanciar el *stub* (que internamente estará utilizando el localizador):

```
HolaMundoIF service = new HolaMundoIFProxy();

System.out.println("Resultado: " + service.saluda("Miguel"));
```

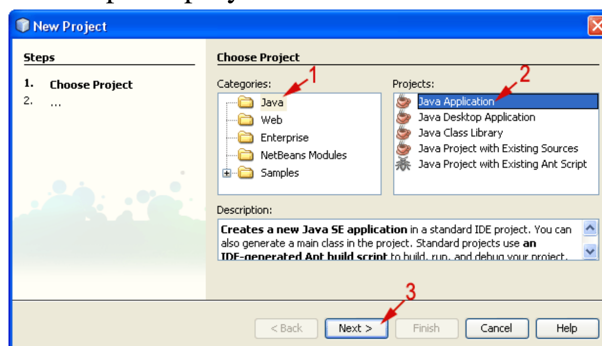
En algunos casos podría interesarnos proteger los servicios con seguridad declarativa, para que no pueda acceder a ellos cualquiera. Dado que los servicios web realmente se invocan mediante un servlet de Axis que está mapeado a una URL, podemos añadir seguridad declarativa para impedir el acceso a dicha URL al igual que se hace para cualquier otro recurso. Para acceder a servicios protegidos mediante seguridad del servidor, deberemos especificar en el cliente el login y el password con los cuales queremos acceder. Esto debe hacerse en el objeto *stub*:

```
HolaMundoMovilLocator locator = new HolaMundoMovilLocator();
HolaMundoIF service = locator.getHolaMundoIFPort();

((HolaMundoIFBindingStub)service).setUsername("usuario");
((HolaMundoIFBindingStub)service).setPassword("password");

System.out.println("Resultado: " + service.saluda("Miguel"));
```

8.2.3. Invocación de servicios web con Netbeans

Vamos a ver ahora cómo crear un cliente de servicios web de forma visual mediante Netbeans. Este entorno utilizará internamente las librerías y herramientas estándar de Sun para crear *stub* del cliente.

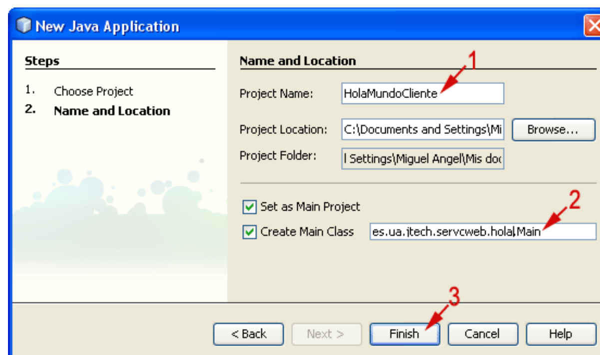
- En primer lugar, deberemos crear un proyecto, o utilizar uno ya existente. Este proyecto podrá ser de cualquier tipo (aplicación Java, aplicación Web, etc). Para nuestro ejemplo crearemos un proyecto Java, aunque podríamos añadir un cliente de servicios a cualquier otro tipo de proyecto de la misma forma.



Creación de un cliente desde Netbeans. Paso 1.

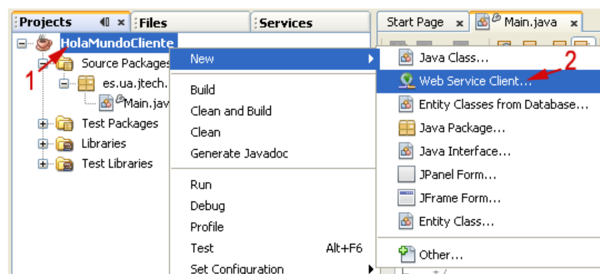
- El proyecto se llamará `HolaMundoCliente` y tendrá una clase principal `Main` en el paquete `es.ua.jtech.servcweb.hola`.

Copyright © 2008-2009 Depto. CCIA All rights reserved.



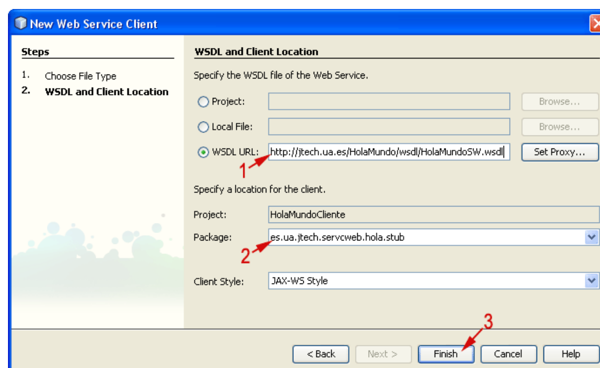
Creación de un cliente desde Netbeans. Paso 2.

- Una vez tengamos un proyecto creado, podremos añadir el *stub* para acceder a un servicio web pulsando con el botón derecho sobre el proyecto y seleccionando la opción *Web Service Client ...*.



Creación de un cliente desde Netbeans. Paso 3.

- Se abrirá un asistente para crear el *stub* del cliente. Aquí deberemos especificar en primer lugar el documento WSDL que define el servicio al que vamos a acceder, por ejemplo indicando la URL en la que se encuentra. En segundo lugar especificaremos el paquete en el que queremos que se generen las clases del *stub*. Podremos también especificar la librería que queremos utilizar: JAX-WS o JAX-RPC. Se añadirán automáticamente las librerías necesarias al proyecto.



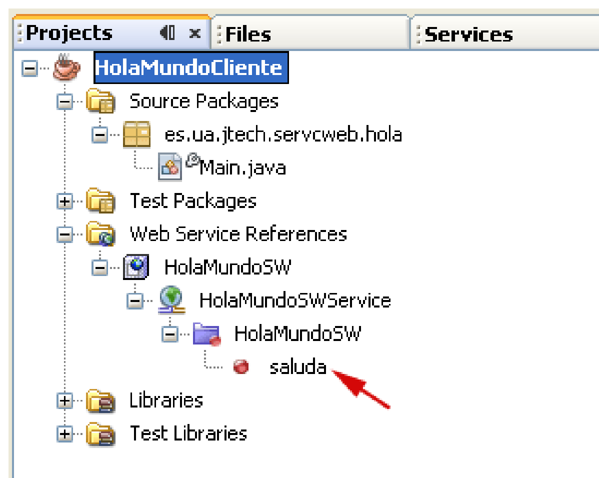
Creación de un cliente desde Netbeans. Paso 4.

Nota:

Copyright © 2008-2009 Depto. CCIA All rights reserved.

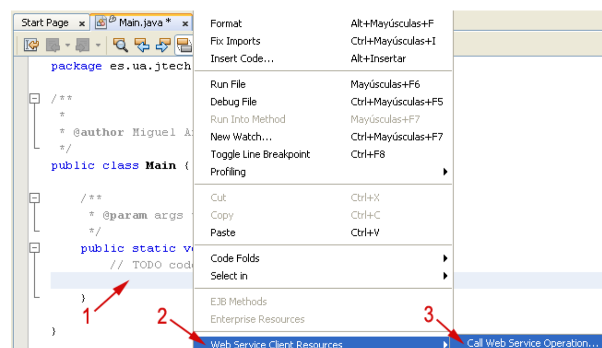
> Una vez seleccionada la librería para acceder a los servicios web dentro de un proyecto, en los sucesivos clientes de servicios que se creen dentro de dicho proyecto siempre se deberá utilizar la misma librería. Es decir, JAX-RPC y JAX-WS no pueden coexistir dentro del mismo proyecto.

- Finalmente, una vez introducidos estos datos pulsaremos el botón *Finish*, tras lo cual Netbeans lanzará la tarea de ant `wsimport` para generar el *stub* del servicio. Una vez generado el *stub* del cliente, podremos ver en la pestaña *Projects*, dentro del apartado *Web Service References*, los datos del servicio al que vamos a acceder y las operaciones que ofrece. Sin embargo, no se muestran las clases Java generadas. Para ver estas clases deberemos ir a la pestaña *Files*, y dentro de ella al directorio `/build/generated/wsimport/client/`.



Creación de un cliente desde Netbeans. Paso 5.

- Una vez creado el *stub* del cliente, deberemos utilizarlo en nuestra aplicación para acceder al servicio. Vamos a suponer que lo queremos invocar desde la clase principal de nuestra aplicación (`Main`), aunque lo mismo podría hacerse para invocarlo desde cualquier otra clase, o incluso desde un JSP. Para crear el código que llame al servicio pulsaremos con el botón derecho del ratón en el lugar del fuente de nuestra clase en el que queramos insertar la llamada (dentro del método `main` en nuestro caso), y seleccionaremos la opción *Web Service Client Resources > Call Web Service Operation*.



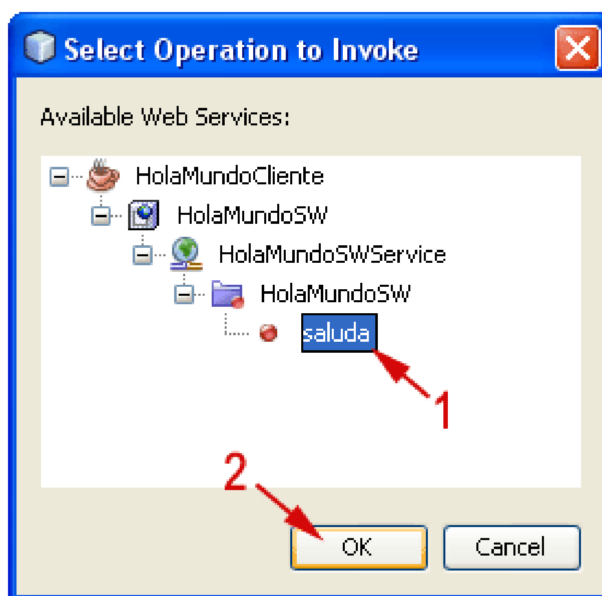
Copyright © 2008-2009 Depto. CCIA All rights reserved.

Creación de un cliente desde Netbeans. Paso 6.

Nota

De forma alternativa podríamos añadir el código que invoca el servicio arrastrando el elemento correspondiente a la operación que queremos invocar desde la ventana de proyectos (*Projects*) hasta el lugar del código donde queremos hacer la invocación.

- Se abrirá una ventana en la que deberemos seleccionar la operación que queramos invocar. En nuestro caso seleccionaremos la operación `saluda` del servicio `HolaMundoSW` y pulsamos el botón *OK*.



Creación de un cliente desde Netbeans. Paso 7.

- Una vez hecho esto se introducirá en nuestra clase el código que hace la llamada al servicio. Deberemos editar este código para especificar los parámetros que vamos a proporcionar en la llamada.

```
    public static void main(String[] args) {
        // TODO code application logic here
        try { // Call Web Service Operation
            es.ua.jtech.servcweb.hola.stub.HolaMundoSWService service = new es.u
            es.ua.jtech.servcweb.hola.stub.HolaMundoSW port = service.getHolaMun
            // TODO initialize WS operation arguments here
            java.lang.String nombre = "Miguel Angel";
            // TODO process result here
            java.lang.String result = port.saluda(nombre);
            System.out.println("Result = " + result);
        } catch (Exception ex) {
        // TODO handle custom exceptions here
        }
    }
}
```

Creación de un cliente desde Netbeans. Paso 8.

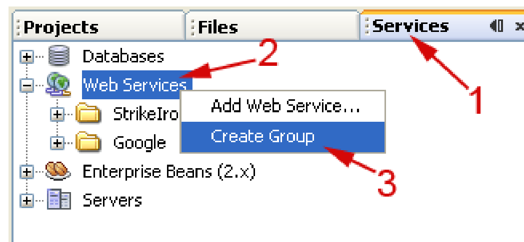
- Con esto ya podremos ejecutar nuestra aplicación cliente, que se conectará al servicio web `HolaMundoSW` para utilizar la operación `saluda`.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

8.2.4. Gestor de servicios web de Netbeans

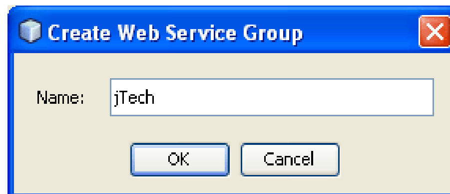
En Netbeans también encontramos un gestor de servicios web, en el que tenemos disponibles algunos servicios web externos proporcionados por terceros (Google, StrikeIron), y al que podemos añadir otros servicios, propios o ajenos. De esta forma podemos tener un catálogo de servicios que podremos incorporar de forma rápida a nuestras aplicaciones, e incluso probarlos desde el propio entorno.

- Podemos acceder a este gestor de servicios a través de la sección *Web Services* de la ventana *Services* de Netbeans. Aquí veremos una lista de servicios que ya vienen incluidos en Netbeans y que podremos probar o incorporar en nuestras aplicaciones. Vamos a añadir un nuevo servicio. Para ello antes creamos un grupo en el que incluirlo. Pulsamos con el botón derecho sobre *Web Services* y seleccionamos *Create Group* para crear el nuevo grupo.



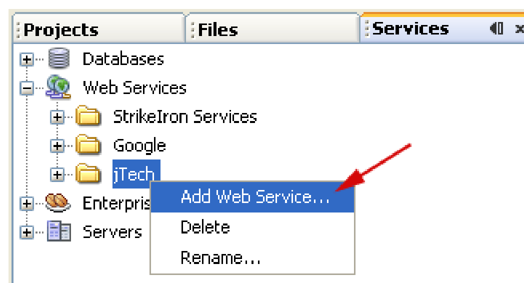
Gestor de servicios web de Netbeans. Paso 1.

- Vamos a añadir un servicio *Hola Mundo* disponible en la web *jTech*. Por lo tanto, crearemos un grupo de nombre *jTech* en el que incluiremos los servicios proporcionados por dicha entidad.



Gestor de servicios web de Netbeans. Paso 2.

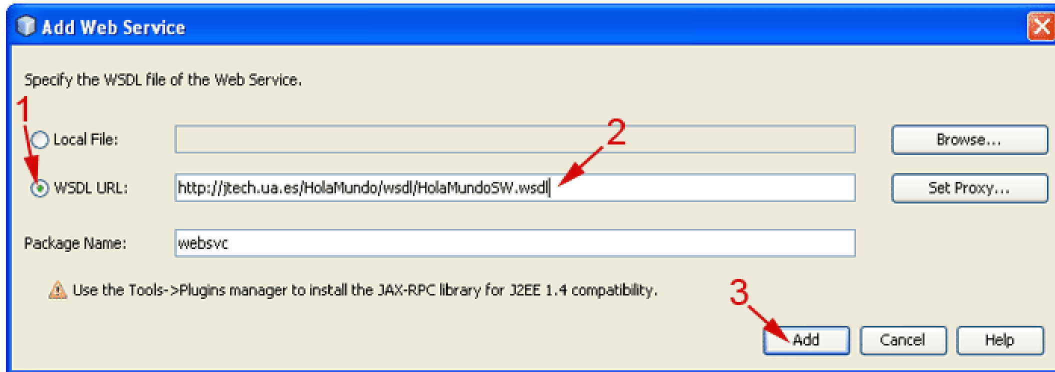
- Pulsamos sobre el grupo con el botón derecho y seleccionamos *Add Web Service ...* para añadir el nuevo servicio.



Gestor de servicios web de Netbeans. Paso 3.

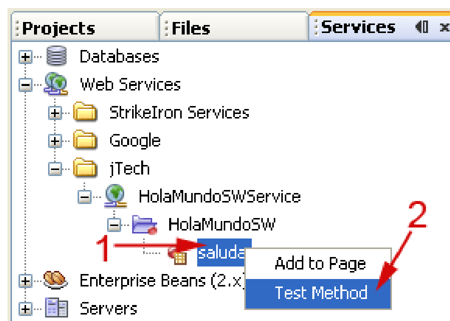
Copyright © 2008-2009 Depto. CCIA All rights reserved.

- Como datos del servicio proporcionamos la dirección del documento WSDL que lo define. Además podemos también especificar el paquete en el cual generará la librería con el *stub* necesario para acceder al servicio. Cuando terminemos de introducir los datos pulsamos *Add* para añadir el nuevo servicio.



Gestor de servicios web de Netbeans. Paso 4.

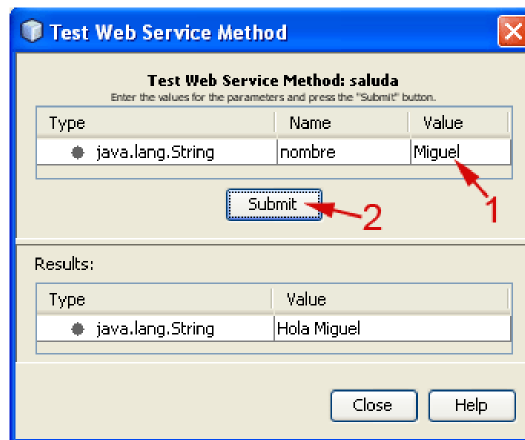
- Una vez tenemos el servicio añadido, podemos desplegarlo para ver las operaciones que ofrece y probarlas pulsando sobre ellas con el botón derecho y seleccionando *Test Method*



Gestor de servicios web de Netbeans. Paso 5.

- Nos aparecerá una ventana en la que deberemos introducir los parámetros necesarios para invocar el servicio, y tras ello pulsar sobre el botón *Submit* para invocarlo y ver el resultado obtenido.

Copyright © 2008-2009 Depto. CCIA All rights reserved.



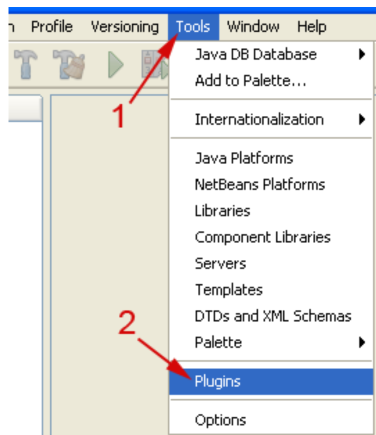
Gestor de servicios web de Netbeans. Paso 6.

También podemos utilizar los servicios del gestor en nuestras aplicaciones de forma sencilla. Simplemente tendremos que arrastrar la operación que queramos invocar sobre el lugar del código fuente en el que queramos hacer la llamada. Al hacer esto se añadirá a nuestro proyecto una librería (JAR) con el *stub* necesario para acceder al servicio.

8.2.5. JAX-RPC en Netbeans

A partir de Netbeans 6 este entorno ya no incluye soporte "de serie" para la librería JAX-RPC. De hecho, lo recomendable es siempre utilizar JAX-WS. Sin embargo, si necesitásemos utilizar servicios web antiguos, de tipo `rpc/encoded`, que no cumplen los estándares actuales, y no tuviesemos la posibilidad de actualizar dichos servicios para hacer que los cumpliesen, entonces será necesario utilizar JAX-RPC para invocarlos.

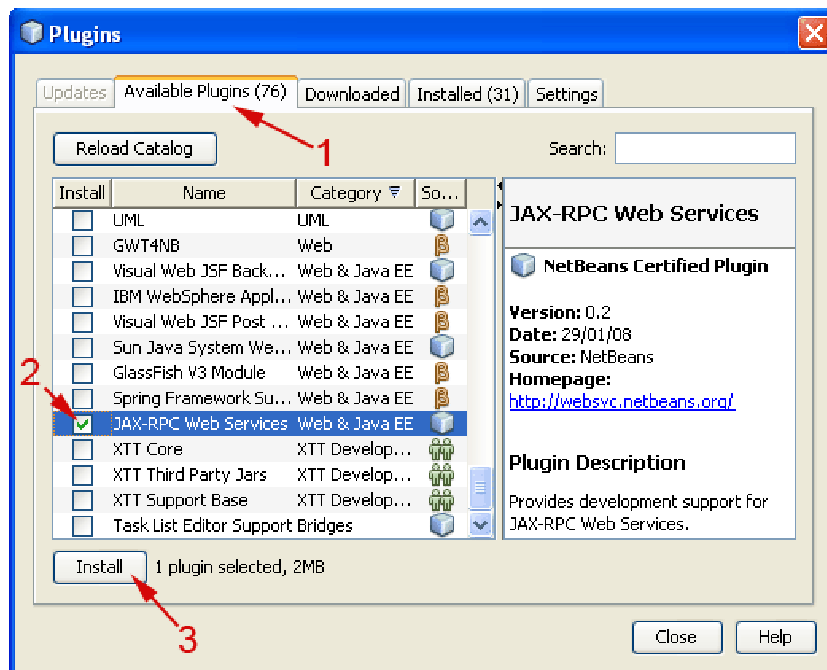
Para poder utilizar JAX-RPC dentro de Netbeans deberemos añadir un *plugin* que incorpore dicha librería. Para instalar el *plugin* pulsaremos sobre la opción *Plugins* del menú *Tools*.



Copyright © 2008-2009 Depto. CCIA All rights reserved.

Instalación de plugins.

Dentro de la ventana de *plugins*, iremos a la pestaña *Available Plugins*, y de la lista de *plugins* disponibles marcaremos *JAX-RPC Web Services*. Tras hacer esto pulsamos el botón *Install* para comenzar la instalación, y seguimos los pasos que nos indica.



Instalar el plugin JAX-RPC.

Los *stubs* generados por JAX-RPC son menos claros que los de JAX-WS, y al añadir el código necesario para invocar el servicio se completan menos datos (no se auto-rellenan ni los parámetros ni el valor de retorno de la operación invocada). Además Netbeans no es capaz de encontrar de forma automática las clases del *stub*, por lo que no puede añadir los `import` necesarios automáticamente. Por lo tanto, resulta recomendable importar de forma manual todas las clases del paquete en el que hayamos generado el *stub*. Por ejemplo:

```
import es.ua.jtech.servcweb.hola.stub.*;
```

De esta forma Netbeans localizará correctamente las clases necesarias y podremos aprovechar la característica de autocompletar para investigar los tipos de datos que se esperan como parámetros, y los tipos de datos que devuelven las funciones invocadas.

8.3. Interfaz de invocación dinámica (DII)

Mediante esta interfaz ya no utilizaremos un *stub* para invocar los métodos del servicio, sino que nos permitirá invocar los métodos de forma dinámica, indicando simplemente el

Copyright © 2008-2009 Depto. CCIA All rights reserved.

nombre del método que queremos invocar como una cadena de texto, y sus parámetros como un *array* de objetos.

Esto nos permitirá utilizar servicios que no conocemos previamente. De esta forma podremos implementar por ejemplo un *broker* de servicios. Un *broker* es un servicio intermediario, al que podemos solicitar alguna tarea que necesitemos. Entonces el *broker* intentará localizar el servicio más apropiado para dicha tarea en un registro de servicios, y lo invocará por nosotros. Una vez haya conseguido la información que requerimos, nos la devolverá. De esta forma la localización de servicios se hace totalmente transparente para nosotros.

Podremos acceder con esta interfaz tanto si contamos con un documento WSDL como si no contamos con él, pero en el caso de que no tengamos el WSDL deberemos especificar en el código todos los datos incluidos en estos documentos que necesitemos y de los que en este caso no disponemos (*endpoint*, parámetros y tipos, etc).

8.3.1. A partir de un documento WSDL

Vamos a ver el caso en el que contamos con el documento WSDL que describe el servicio. El primer paso será conseguir el objeto `Service` igual que hicimos en el caso anterior:

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

Utilizaremos el objeto `Call` para hacer las llamadas dinámicas a los métodos del servicio. Deberemos crear un objeto `Call` correspondiente a un determinado puerto y operación de nuestro servicio:

```
Call call = serv.createCall(
  new QName("http://jtech.ua.es", "HolaMundoPortTypeSoapPort"),
  new QName("http://jtech.ua.es", "saluda"));
```

El último paso será invocar la llamada que hemos creado:

```
Integer result = (Integer) call.invoke(
                 new Object[] { "Miguel" });
```

A este método le debemos proporcionar un *array* de objetos como parámetro, ya que debe poder utilizarse para cualquier operación, con diferente número y tipo de parámetros. Como tampoco se conoce *a priori* el valor devuelto por la llamada, deberemos hacer una conversión *cast* al tipo que corresponda, ya que nos devuelve un `Object` genérico.

8.3.2. Sin un documento WSDL

Si no contamos con el WSDL del servicio, crearemos un objeto `Service` proporcionando únicamente el nombre del servicio:

Copyright © 2008-2009 Depto. CCIA All rights reserved.

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

A partir de este objeto podremos obtener un objeto `Call` para realizar una llamada al servicio de la misma forma que vimos en el caso anterior:

```
Call call = serv.createCall(
  new QName("http://jtech.ua.es", "HolaMundoPortTypeSoapPort"),
  new QName("http://jtech.ua.es", "saluda"));
```

En este caso el objeto `Call` no tendrá ninguna información sobre las características del servicio, ya que no tiene acceso al documento WSDL que lo describe, por lo que deberemos proporcionárselas nosotros explícitamente.

En primer lugar, deberemos especificar el *endpoint* del servicio, para que sepa a qué dirección debe conectarse para acceder a dicho servicio:

```
call.setTargetEndpointAddress(endpoint);
```

Una vez especificada esta información, deberemos indicar el tipo de datos que nos devuelve la llamada a la operación que vamos a invocar (en nuestro ejemplo `saluda`):

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

Por último, indicaremos los parámetros de entrada que toma la operación y sus tipos:

```
QName t_double =
  new QName("http://www.w3.org/2001/XMLSchema", "string");
call.addParameter("string_1", t_string, ParameterMode.IN);
```

Una vez hecho esto, podremos invocar dicha operación igual que en el caso anterior:

```
Integer result = (Integer) call.invoke(
                 new Object[] { "Miguel" });
```


Copyright © 2008-2009 Depto. CCIA All rights reserved.

Copyright © 2008-2009 Depto. CCIA All rights reserved.