Departamento de Ingeniería Telemática
Escuela Superior de Ingenieros
Universidad de Sevilla

PRÁCTICA 9.

Servicios web

1. OBJETIVO

El objetivo de esta práctica es desarrollar un ejemplo básico de servicio Web, así como conocer algunas de las tecnologías implicadas en el desarrollo de servicios Web (SOAP, WSDL y UDDI). También se construye un cliente de servicio Web remoto conocida la interfaz WSDL de dicho servicio Web. Para la realización de la práctica se utiliza un software basado en Java de libre distribución: Axis, de la Apache Software Foundation.

2. DESCRIPCIÓN

Los servicios Web (Web Services, WS) están pensados para dar soporte a interacciones entre máquinas. Para que estas interacciones sean posibles todo servicio Web posee una interfaz descrita en un formato procesable por una máquina (utilizando el lenguaje WSDL, Web Service Description Language). En la descripción del WS se indica cómo otros sistemas pueden interactuar con el WS: qué mensajes se intercambian, qué parámetros deben pasarse al servicio para que lleve a cabo su función, qué resultados se obtienen o qué fallos pueden producirse. Normalmente la invocación remota a un WS se lleva a cabo utilizando mensajes SOAP (Simple Object Accesss Protocol), y transmitidos mediante el protocolo HTTP (HyperText Transfer Protocol). Para facilitar la localización de servicios, es posible registrar los mismos en un directorio o registro UDDI (Universal Description, Discovery and Integration). La descripción del servicio almacenada en el registro UDDI contiene información como el nombre del servicio, detalles sobre el proveedor del servicio, el tipo de servicio ofrecido, su ubicación (URL) y detalles técnicos sobre el mismo (el WSDL del servicio). La información de esta descripción es utilizada por las aplicaciones cliente para decidir qué servicios pueden invocar.

Los Web Services son una tecnología middleware adecuada cuando se pretenden desarrollar arquitecturas orientadas a servicio (SOA, Service Oriented Architecture). En estas arquitecturas distintos componentes interactúan intercambiando mensajes para conseguir llevar a cabo una tarea compleja. Gracias a la utilización de WS, que pueden ser accedidos remotamente, componentes distintos pueden ejecutarse en distintas máquinas distribuidas por toda la red. Actualmente este tipo de arquitectura goza de gran acogida en entornos B2B (Business to Business).

2.1 Instalación del entorno Apache Axis

Axis está formado por un conjunto de librerías Java que nos permitirán construir procesadores SOAP para ser utilizados en la implementación de clientes de servicios Web, pero también para desarrollar nuestros propios servicios o incluso pasarelas, *gateways*, que mediarán en la invocación a servicios. Para facilitar esta tarea de desarrollo, Axis proporciona algunas facilidades como son por ejemplo:

- Un sencillo servidor Web para los servicios que desarrollemos.
- Herramientas de conversión entre Java y WSDL.
- Herramientas de depuración como un monitor de conexiones TCP.

Asimismo, Axis proporciona los mecanismos necesarios para que los servicios Web ya desarrollados puedan ser fácilmente instalados en servidores más avanzados que el que Axis proporciona, como son el Apache Tomcat.

⇒ 1. Descargue y descomprima el fichero correspondiente a esta práctica.

Para instalar el entorno Axis en su máquina descomprima el fichero dado con los ficheros de esta práctica que lleva por nombre **axis-bin-1_4.tar.gz**. Descomprima dicho fichero utilizando el comando **tar xvzf axis-bin-1_4.tar.gz**. Como resultado debería aparecer un directorio **axis-1_4** conteniendo otra serie de directorios con la distribución de Axis. Como todo entorno basado en Java, para poder utilizar desde nuestros programas las funcionalidades ofrecidas por las librerías de Axis, será necesario incluir en el ***classpath*** los ficheros jar de dichas librerías. En concreto será necesario añadir al *classpath* los siguientes ficheros:

- Los ficheros jar que se encuentran en el directorio **lib** de la distribución de Axis: axis-ant.jar, commons-logging-1.0.4.jar, axis.jar, jaxrpc.jar, saaj.jar, commons-discovery-0.2.jar, log4j-1.2.8.jar y wsdl4j-1.5.1.jar.
- El fichero dado mail.jar
- El fichero dado activation.jar

Recuerde que para añadir los ficheros jar al *classpath*, puede modificar la variable de entorno **CLASSPATH** o simplemente utilizar la opción **-classpath** de los comandos *java* y *javac*.

NOTA: Para la compilación y ejecución de los ejemplos, se proporciona un fichero makefile con la opción –cp (-classpath) del compilador adecuada para ello. Se proporciona también un fichero llamado notas.txt con las órdenes para la ejecución de los procesos de axis que se indican en los ejemplos.

2.2 El servicio Web HolaMundo

En primer lugar se implementará y probará un servicio básico, posteriormente añadiremos funcionalidades más complejas. Como suele ser habitual, el servicio escogido es el **HolaMundo**, que constará de un único método invocable remotamente y denominado **hola**. Dicho método recibirá como parámetro una cadena de texto con un nombre y devolverá la cadena de texto "Hola " + nombre recibido. Una implementación en Java de este servicio podría ser la siguiente:

```
public class HolaMundo {

        public String hola(String nombre) {

            return new String("Hola " + nombre + "!");

        }
    }
```

HolaMundo.java

La forma más sencilla de instalar y poner a funcionar un servicio Web en Axis es la que se llama ***Instant Deployment*** o instalación instantánea. Esta forma no requiere ninguna tarea adicional a la de generar la implementación Java del servidor. Será suficiente con que almacenemos la implementación en un fichero de **extensión jws**, *Java Web Service* y hagamos ese fichero accesible al servidor. En el caso del servidor proporcionado con el sistema Axis, para hacer el fichero jws accesible al mismo basta con copiarlo en el directorio de aplicaciones de dicho servidor, denominado **webapps/axis**. Así pues, almacene el código fuente que se proporciona del servicio HolaMundo en un fichero **HolaMundo.jws** y copie dicho fichero al directorio **$AXIS_HOME/webapps/axis**, donde **$AXIS_HOME** debe ser sustituido por el directorio donde está instalado el sistema Axis en su cuenta.

Una vez finalizada la instalación del código del servidor, será necesario codificar el cliente que invocará al método hola del servicio HolaMundo. Un ejemplo de cliente para hacer esto es el siguiente:

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;

import javax.xml.rpc.ParameterMode;

public class HolaMundoClient
{
  public static void main(String[]args) throws Exception
  {

    Options options = new Options(args);

    String endpoint = "http://localhost:" + options.getPort() +
      "/axis/HolaMundo.jws";

    Service service = new Service();

    Call call = (Call) service.createCall();

    call.setTargetEndpointAddress(new java.net.URL(endpoint));
    call.setOperationName("hola");
    call.addParameter("nombre", XMLType.XSD_STRING, ParameterMode.IN);
    call.setReturnType(XMLType.XSD_STRING);

    String ret = (String) call.invoke(new Object[]{ "Axis" });

    System.out.println(ret);
```

```
    }
}
```

HolaMundoClient.java

Este cliente recibe como parámetro el puerto en el que se encuentra escuchando el servidor a invocar y realiza las siguientes operaciones:

- Captura los parámetros de entrada al cliente en un objeto Options.
- Calcula la ubicación del servidor a invocar (*endpoint*) teniendo en cuenta el puerto pasado por parámetro.
- Construye un objeto Service y un objeto Call que representan el servicio a invocar y una llamada o invocación a ese servicio.
- Con el método *setTargetEndpointAddress* se establece el servidor a llamar.
- Con el método *setOperationName* se indica el método a invocar.
- Con el método *addParameter* se establece un parámetro a pasar al servidor, indicando su nombre, su tipo (utilizando los tipos de la especificación XML Schema) y el modo de parámetro, en este caso se trata de un parámetro de entrada.
- Con el método *setReturnType* se indica el tipo esperado como resultado.
- Con el método *invoke* se lleva a cabo la invocación remota.
- Y finalmente se imprime el resultado recibido.

⇒ 2. Compile el código fuente del cliente (no se olvide añadir las librerías Axis al *classpath*). Active el servidor Axis para poner nuestro servicio Web a la escucha haciendo **desde el directorio $AXIS_HOME/webapps**:

```
java org.apache.axis.transport.http.SimpleAxisServer -p [puerto]
```

Utilizando por ejemplo como **puerto** el **8888** y lance el cliente para invocar el servicio Web de la siguiente forma:

```
java HolaMundoClient -p [puerto]
```

Donde el parámetro puerto que se le pasa al servidor Axis y el que se le pasa al cliente del servicio Web, deben coincidir.

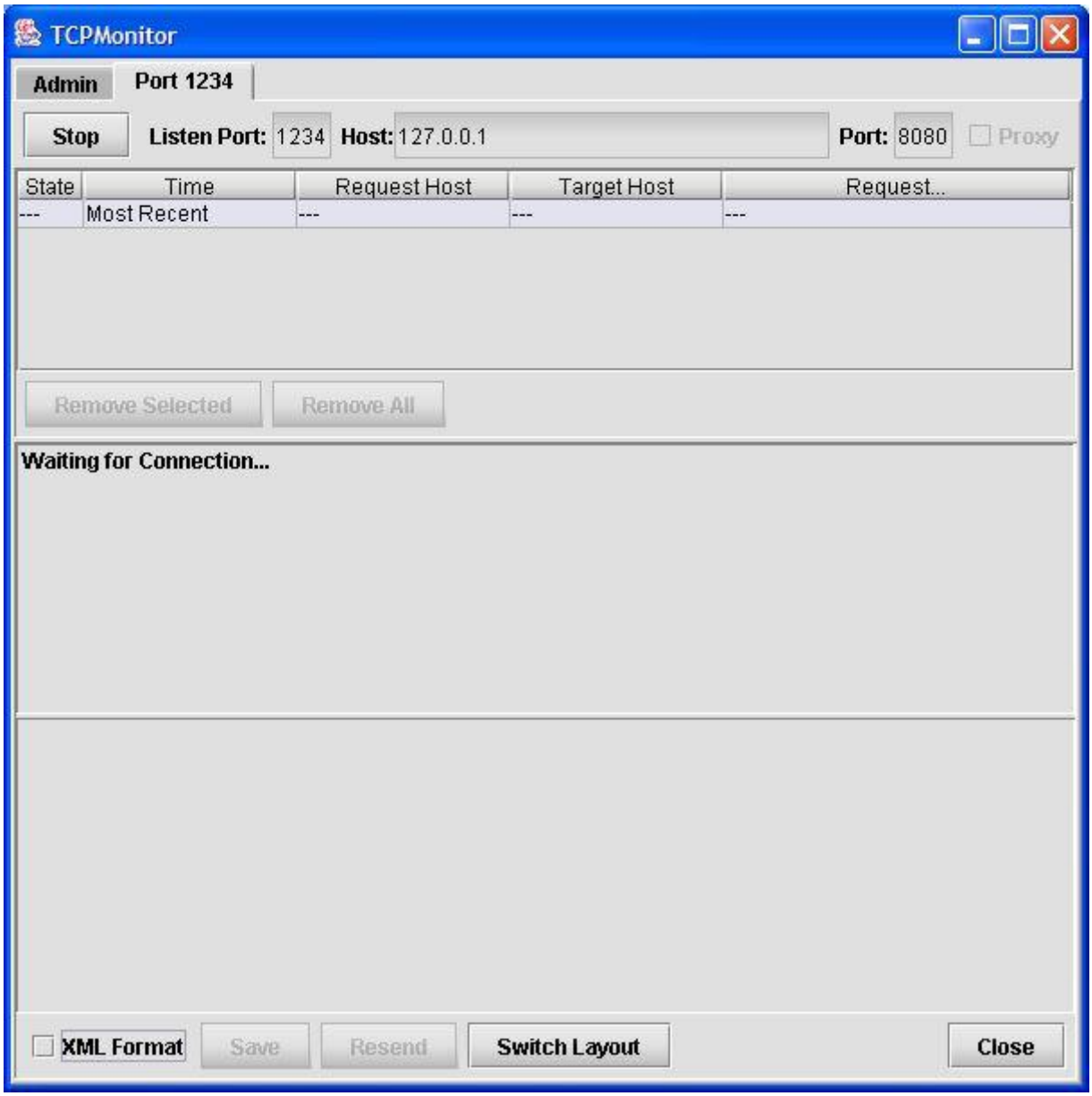
Si por algún motivo se desea detener el servidor de Axis, ejecutaremos:

```
java org.apache.axis.client.AdminClient -p [puerto] quit
```

2.3 Usando el monitor TCP

Como ya se indicó anteriormente, el entorno Apache Axis proporciona herramientas adicionales para la depuración de servicios Web. Una de estas herramientas es el monitor de conexiones TCP, *tcpmon*. El monitor de conexiones TCP no es más que una herramienta gráfica que nos permitirá observar qué información concreta se intercambian el cliente y el servidor del servicio Web, para permitirnos detectar visualmente posibles irregularidades. Básicamente se trata de una pasarela que recibe las peticiones de los clientes, imprime la información HTTP/SOAP en

pantalla (el contenido de los paquetes TCP, de ahí su nombre) y redirige la solicitud al servidor, capturando sus resultados y mostrándolos igualmente en pantalla antes de enviarlos al cliente.



Como se puede apreciar en la figura anterior, la parte inferior de la ventana del monitor está dividida en dos secciones, una superior en la que podremos visualizar la información que fluye en la dirección cliente-servidor y otra inferior en la que visualizaremos el flujo contrario de información, servidor-cliente.

⇒ 3. En este ejercicio, usaremos el monitor TCP para visualizar el flujo de información resultante de la comunicación entre cliente y servidor del ejercicio anterior. Para ello lleve a cabo las siguientes operaciones:

- Lanzar el monitor TCP de Axis, utilizando:

  java org.apache.axis.utils.tcpmon 9999 localhost 8888

- Lanzar el servidor básico de axis, haciendo **desde el directorio $AXIS_HOME/webapps**:

  java org.apache.axis.transport.http.SimpleAxisServer -p 8888

- Lanzar el cliente *HolaMundoClient*, haciendo:

  java HolaMundoClient -p 9999

Observe que el cliente redirige su petición al puerto 9999, mientras que el servidor está escuchando en el puerto 8888. Eso es así porque es el monitor TCP, el que está escuchando en el puerto 9999 y dirigiendo las peticiones del cliente al puerto 8888 de *localhost*.

2.4 Excepciones remotas

Durante la invocación a un método remoto, pueden producirse en el servidor situaciones excepcionales que obliguen a devolver errores al cliente. Uno de los mecanismos más utilizados en el lenguaje Java para indicar que se ha producido una situación anómala es mediante el uso de excepciones.

⇒ 4. En este ejercicio utilizaremos el monitor TCP para ver cómo se gestionan en SOAP las excepciones Java lanzadas por el código servidor. Para ello se pide que implemente un servicio Web de nombre **Conversor** que constará de un único método **convierte** que recibirá como parámetro una cadena de texto y devolverá un entero con el número representado por la cadena. Para hacer la conversión cadena-entero puede utilizar el método **parseInt** de la clase Integer. Si observa el API de Java, verá que este método devuelve excepciones en caso de que la cadena recibida como parámetro no represente un número entero. Para tratar esas excepciones declare el método **convierte** de tal forma que las lance hacia el cliente. Allí, se deberán capturar las excepciones que puedan surgir al hacer la invocación remota a la manera tradicional de Java (**try{...} catch(...) {...}**).

Codifique el cliente y el servidor, instale el servidor utilizando el método hasta ahora visto de instalación instantánea y observe, utilizando el monitor TCP, qué es lo que pasa cuando se produce una excepción en el servidor. Pruebe a pasar como parámetros a **convierte** un número entero y una cadena de texto no numérica. Para facilitar esta tarea se aconseja que su cliente reciba como parámetro de línea de comandos la cadena a enviar al servidor.

2.5 Tipos de datos propios

La instalación instantánea que hemos utilizado en los apartados anteriores está pensada para servicios muy básicos y no permite por ejemplo trabajar con paquetes Java o definir tipos propios. Para servicios más avanzados será necesario emplear lo que se llama ***Custom Deployment*** o instalación personalizada. En esta sección presentaremos esta instalación personalizada mediante el desarrollo de un servicio Web que actuará como proveedor de noticias. El código fuente del servidor del servicio Web se proporciona en el directorio newswebservice. La clase que implementa el servidor es la clase **Periodico** cuyo código puede ver a continuación:

```java
package newswebservice;

import java.util.Vector;


public class Periodico {

    private Vector noticias = null;

    public Periodico() {

      noticias = new Vector();

    }

    public void insert(Noticia n) throws Exception {

      if (n != null)
          noticias.add(n);
      else
          throw new Exception("Noticia invalida!");

    }

    public Noticia[] query(String texto) throws Exception {

      Vector v = new Vector();

      for (int i=0; i < noticias.size(); i++) {

          Noticia n = (Noticia)noticias.get(i);

          if (n.getTitular().indexOf(texto) != -1) {
            v.add(n);
          }
      }

      if (v.size() > 0) {

          Noticia vn[] = new Noticia[v.size()];

          for (int k=0; k < v.size(); k++) {
            vn[k] = (Noticia)v.get(k);
          }

          return vn;

      }
      else
          throw new Exception(texto + " no encontrado.");

    }

}
```

Periodico.java

Se ha definido un paquete **newswebservice** y adicionalmente a la clase **Periodico** que implementa el servidor, se necesita una clase **Noticia** que es un tipo propio utilizado para representar noticias. El código correspondiente a la clase **Noticia** es el siguiente:

```
package newswebservice;

public class Noticia implements java.io.Serializable {

    private String titular;
    private String descripcion;
    private String url;

    public void setTitular(String titular) {
      this.titular = titular;
    }
    public void setDescripcion(String descripcion) {
      this.descripcion = descripcion;
    }
    public void setUrl(String url) {
      this.url = url;
    }

    public String getTitular() {
      return this.titular;
    }
    public String getDescripcion() {
      return this.descripcion;
    }
    public String getUrl() {
      return this.url;
    }

}
```

Noticia.java

Si observa el código del servidor, verá que éste proporciona básicamente dos métodos: uno que permite insertar nuevas noticias en nuestra base de datos, **insert**, y uno que permite consultar dicha base de datos para obtener noticias en cuyo titular se mencione un determinado texto, **query**.

Además del código del servidor, el código fuente del cliente del servicio Web se proporciona en el fichero PeriodicoClient.java cuyo código se muestra a continuación:

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;

import javax.xml.rpc.ParameterMode;
import javax.xml.namespace.QName;

import newswebservice.*;
```

```
public class PeriodicoClient
{

    static String endpoint = "http://localhost:8888/axis/services/Periodico";

    public static void main(String [] args) throws Exception {

      String titular;
      String descripcion;
      String url;

      Noticia n;

      if (args[0].equals("insert")) {
          if (args.length == 4) {

            titular = args[1];
            descripcion = args[2];
            url = args[3];

            n = new Noticia();
            n.setTitular(titular);
            n.setDescripcion(descripcion);
            n.setUrl(url);

            try {
                invoca_insert(n);
                System.exit(0);
            }
            catch (Exception ex) {
                System.out.println(ex);
                System.exit(1);
            }
          }
          else {
            System.out.println("Error en el paso de parametros");
            System.exit(1);
          }
      }


      if (args[0].equals("query")) {
          if (args.length == 2) {
            try {
                invoca_query(args[1]);
                System.exit(0);
            }
            catch (Exception ex) {
                System.out.println(ex);
                System.exit(1);
            }
          }
          else {
            System.out.println("Error en el paso de parametros");
            System.exit(1);
          }
      }

      System.out.println("Error en el paso de parametros");
      System.exit(1);
```

```
    }


    private static void invoca_insert(Noticia n) {

      try {

          Service  service = new Service();
          Call     call    = (Call) service.createCall();
          QName    qn       = new QName( "http://www.uc3m.es/WS/Periodico",
"Noticia" );


          call.registerTypeMapping(newswebservice.Noticia.class, qn,
                                   new
org.apache.axis.encoding.ser.BeanSerializerFactory(newswebservice.Noticia.cla
ss, qn),
                                   new
org.apache.axis.encoding.ser.BeanDeserializerFactory(newswebservice.Noticia.c
lass, qn));

          call.setTargetEndpointAddress( new java.net.URL(endpoint) );
          call.setOperationName("insert");
          call.addParameter("n", qn, ParameterMode.IN);
          call.setReturnType(XMLType.AXIS_VOID);
          call.invoke(new Object [] { n });

      }
      catch (Exception ex) {

          System.out.println(ex);
      }

    }


    private static void invoca_query(String texto) {

      try {

          Service  service = new Service();
          Call     call    = (Call) service.createCall();
          QName    qn      = new QName( "http://www.uc3m.es/WS/Periodico",
"Noticia" );
          QName    qna     = new QName( "http://www.uc3m.es/WS/Periodico",
"ArrayOfNoticia" );

          call.registerTypeMapping(newswebservice.Noticia.class, qn,
                                   new
org.apache.axis.encoding.ser.BeanSerializerFactory(newswebservice.Noticia.cla
ss, qn),
                                   new
org.apache.axis.encoding.ser.BeanDeserializerFactory(newswebservice.Noticia.c
lass, qn));

          call.setTargetEndpointAddress( new java.net.URL(endpoint) );
          call.setOperationName("query");
          call.addParameter("texto", XMLType.XSD_STRING, ParameterMode.IN );

          call.setReturnType(qna);
```

```
            Noticia obj[] = (Noticia [])call.invoke(new Object [] { texto });

            for (int k=0; k < obj.length; k++) {

              Noticia n = obj[k];

              System.out.println("Titular: " + n.getTitular());
              System.out.println("Descripcion: " + n.getDescripcion());
              System.out.println("URL: " + n.getUrl());
            }

        }
        catch (Exception e) {

            System.out.println(e);
        }
      }

}
```

PeriodicoClient.java

Analice el código fuente proporcionado observando aspectos como:

- Cómo se le proporciona un nombre único, un identificador, al tipo Noticia mediante la creación de un objeto QName (*Qualified Name* o nombre cualificado).
- Cómo se gestiona el caso en el que el método remoto no devuelva información (tipo *void*).
- Cómo es necesario registrar el tipo Noticia con el método **registerTypeMapping**. Observe que la clase Noticia se comporta como un *Bean* de ahí que estemos utilizando los serializadores y deserializadores por defecto para *Beans*. Si nuestro tipo lo requiriese se podrían especificar clases propias como serializadores/deserializadores.
- Observe como adicionalmente al tipo noticia se declara en el método **invoca_query** un tipo al que se le da el nombre cualificado *http://www.uc3m.es/WS/Periodico#ArrayOfNoticia* para gestionar la recepción de un array de noticias en el cliente como resultado de la invocación al método **query** del servidor.

Una vez ha analizado el código fuente del cliente y del servidor, veremos cómo se instala nuestro servicio Web utilizando la técnica del ***Custom Deployment***. Para ello será necesario generar un fichero de configuración XML con toda la información necesaria para la instalación. Dicho fichero es lo que se denomina un**Web Service Deployment Descriptor (WSDD)**, o descriptor de instalación de servicio Web, y se suele almacenar en un fichero de extensión **wsdd**. En nuestro caso el código fuente del fichero de instalación será el siguiente:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
                xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="Periodico" provider="java:RPC">

     <parameter name="scope" value="application"/>
      <parameter name="className" value="newswebservice.Periodico"/>
      <parameter name="allowedMethods" value="*"/>
```

```
      <beanMapping qname="ns:Noticia"
                   xmlns:ns="http://www.uc3m.es/WS/Periodico"
                   languageSpecificType="java:newswebservice.Noticia"/>

      <arrayMapping qname="ns:ArrayOfNoticia"
                    xmlns:ns="http://www.uc3m.es/WS/Periodico"
                    languageSpecificType="java:newswebservice.Noticia[]"
                    innerType="ns:Noticia"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

   </service>

</deployment>
```

---

deploy.wsdd

Básicamente en este fichero se proporciona la siguiente información:

- El nombre del servicio (*Periodico*) y su tipo (en nuestro caso será siempre *java:RPC*).
- El ámbito o *scope* del servicio. Básicamente existen tres posibles valores: **request**, **session** y **application**. Esto indica el tiempo de vida del objeto que proporciona el servicio: si sólo dura el tiempo que dura una invocación el valor será *request*, si permanece activo toda la sesión del cliente será *session*, y si permanece vivo todo el tiempo que el servidor Web permanezca activo será *application*.
- El nombre de la clase que presta el servicio *newswebservice.Periodico*.
- Qué métodos de esa clase son invocables remotamente vía SOAP (en este caso, todos los públicos).
- Por último definimos dos tipos propios: uno para las noticias de nombre cualificado *http://www.uc3m.es/WS/Periodico#Noticia* y clase *newswebservice.Noticia* y otro para un array de noticias de nombre cualificado *http://www.uc3m.es/WS/Periodico#ArrayOfNoticia* y tipo *newswebservice.Noticia[]*. Observe que estos nombres coinciden con los que se han utilizado en el código fuente del cliente, de forma que cliente y servidor utilicen los mismos nombres de tipos y puedan entenderse en su comunicación.

Una vez tenemos toda la información que necesitamos para proceder a la instalación del servicio, es necesario llevar a cabo los siguientes pasos para que ésta se lleve a cabo correctamente:

- Generar un fichero jar que contenga todas las clases necesarias para el correcto funcionamiento del servidor. En nuestro caso deberán incluirse las clases **Periodico** y **Noticia**. No olvide que dichas clases pertenecen a un paquete de nombre **newswebservice**, con lo cual deben almacenarse en un directorio con ese nombre, compilarse y generar el jar de todo el directorio utilizando un comando:

  jar cvf newswebservice.jar newswebservice/*.class

- Lanzar el servidor asegurándose de incluir en el *classpath* el fichero **newswebservice.jar**. Para ello se puede copiar dicho fichero en el directorio**$AXIS_HOME/webapps** y ejecutar desde ese mismo directorio:

java -classpath $CLASSPATH:newswebservice.jar:.
org.apache.axis.transport.http.SimpleAxisServer -p 8888

Donde en la variable CLASSPATH deben haberse añadido previamente las librerías de Axis.

- Instalar el servidor utilizando la herramienta de administración de Axis:

  java org.apache.axis.client.AdminClient -p 8888 deploy.wsdd

  Donde el fichero **deploy.wsdd** debe contener el descriptor de instalación del fichero antes proporcionado. Si todo se ejecuta correctamente debería aparecer un mensaje de la forma:

  Processing file deploy.wsdd

  <Admin>Done processing</Admin>

- Ejecutar el código del cliente. Básicamente este recibe un primer parámetro que indica qué comando remoto se va a ejecutar: **query** o **insert**. Los siguientes parámetros son los que cada comando necesita para poder hacer la solicitud al servidor. En el caso del método **query** será la cadena de texto a buscar. En el caso del método **insert** serán el titular, la descripción y una URL que son necesarias para construir un objeto Noticia y enviarlo al servidor.

⇒ 5. Pruebe el funcionamiento de la aplicación proporcionada, siguiendo los pasos especificados anteriormente.

2.6 Usando WSDL

Como hemos visto en la introducción teórica de la práctica, el lenguaje WSDL permite describir la interfaz que un servicio Web ofrece a las aplicaciones remotas. Una situación frecuente cuando se trabaja con servicios Web, es que se desarrolle únicamente una parte del sistema: sólo el cliente o sólo el servidor, siendo terceras partes las que desarrollan la otra parte. Para que esto sea posible, es interesante poder disponer de la especificación en WSDL del servicio, puesto que si estamos desarrollando el cliente nos interesará saber qué operaciones permite el servicio y si desarrollamos el servidor, deberemos publicar nuestro WSDL para facilitar a terceras partes el desarrollo de sus clientes. En este ejercicio veremos cómo obtener la descripción WSDL de un servicio a partir de su interfaz Java y cómo a partir de una interfaz WSDL conocida podremos desarrollar clientes para un determinado servicio.

El proceso de generación de WSDL a partir del código Java de un servicio Web es absolutamente automático en Axis. De hecho, al mismo tiempo que se instala un servicio (en la modalidad de ***Custom Deployment***) se genera automáticamente la descripción WSDL de dicho servicio. Esta descripción puede ser accedida vía Web en la URL **http://host:puerto/axis/services/Nombre_de_Servicio?wsdl**. Por ejemplo, en el caso del servicio Periodico que hemos creado e instalado en el ejercicio anterior, podemos visualizar su descripción WSDL sin más que abrir en el navegador la URL:**http://localhost:8888/axis/services/Periodico?wsdl** (siempre que el servidor Web esté activo, y esté corriendo en el *host* local y el puerto 8888).

Para el proceso inverso, generación de código Java a partir de la descripción WSDL de un servicio, Axis también proporciona herramientas de apoyo a los desarrolladores. En concreto existe un compilador denominado **WSDL2Java** que permite generar automáticamente *stubs* y otros ficheros muy útiles para la codificación de clientes. Pruebe el funcionamiento de esta herramienta sobre el fichero WSDL del servicio Web periódico ejecutando:

java org.apache.axis.wsdl.WSDL2Java "http://localhost:8888/axis/services/Periodico?wsdl"

Observe los ficheros y directorios que se generan. Básicamente el compilador genera por cada tipo una clase Java, además de una clase adicional si el tipo se utiliza como parámetro *OUT* o *INOUT*. Por cada entrada **wsdl:portType** del fichero WSDL se genera una interfaz Java. Por cada entrada **wsdl:binding** del WSDL se genera un *stub*. Por último, por cada entrada **wsdl:service** se genera una interfaz Java y un objeto *locator* que nos permitirá obtener instancias de *stubs* que utilizaremos para invocar los servicios remotos.

Una vez tenemos todos los ficheros auxiliares generados, simplemente será necesario que codifiquemos en un fichero Java un método *main* que haciendo uso de esos objetos intermedios nos permita invocar a los métodos remotos del servicio Web: el cliente. En nuestro caso el código fuente de dicho fichero se proporciona a continuación:

```
// Este fichero usa las clases generadas por axis a partir del WSDL

import es.uc3m.www.WS.Periodico.*;
    import ubuntu.axis.services.Periodico.*;

    public class PeriodicoClient2 {

      public static void main(String [] args) throws Exception {

        // Crear un servicio periodico
        PeriodicoService service = new PeriodicoServiceLocator();

        // Obtener un stub que utilizaremos para invocar los métodos remotos
        Periodico port = service.getPeriodico();

        // Invocar los métodos
        Noticia n = new Noticia("Descripcion1","Titular1","URL1");
        port.insert(n);

        Noticia vn[] = port.query("Titular");

        System.out.println(vn[0].getTitular());
      }
    }
```

PeriodicoClient2.java

Observe que los paquetes que se importan al principio del código dependen de los directorios que ha generado el compilador (que a su vez dependen de la información del WSDL), así que en su caso la ruta de estos paquetes puede ser distinta a la proporcionada.

⇒ 6. Compile y ejecute el cliente generado a partir del wsdl

2.7 Referencias

API de Apache Axis
Guía de usuario de Apache Axis

⇒ 7. Cree un servicio web con una agenda de teléfono. La agenda será un conjunto de teléfonos. Cada teléfono tiene un nombre, un tipo y un número de teléfono (de tipo String).

⇒ 8. En la tarea habilitada para la práctica, entregue un fichero comprimido (.zip) con nombre su login que sea el directorio de nombre su login comprimido (que contiene el directorio P9codigo con todo su trabajo).