Capítulo 6

Pruebas realizadas

6.1 Prueba 1: Números primos

El primer servicio web XML que probaremos en la plataforma montada será muy simple. Consiste en un cliente que pida al usuario que introduzca un número entero; dicho número se enviará al servidor que lo procesará para averiguar si es primo o no, y devuelve la respuesta en un mensaje de texto al cliente, que la muestra por pantalla.

Este servicio servirá como primer contacto con los servicios web XML y como prueba para comparar el rendimiento de JSR 172 y de kSOAP frente a mensajes SOAP muy cortos.

6.1.1 Servidor

El servidor es tan sólo un método remoto desplegado como un servicio web. La creación y la interpretación de los mensajes SOAP es tarea de Axis, siendo totalmente transparente a ésta función.

El servicio es universal y puede ser accedido por cualquier tipo de cliente que conozca las características de la comunicación y que soporte el protocolo SOAP.

6.1.1.1 Servicio

La clase se llama *Primo.java* y contiene el método *esPrimo*, que recibe como parámetro un número entero en formato String y devuelve una cadena de caracteres informando del resultado de la operación.

```
public class Primo {
    public String esPrimo(String a) {
```

En primer lugar definimos las variables: *b* será el parámetro en tipo *double*, necesario para hallar su raíz cuadrada. El entero *num* será un contador, *resultado* será la cadena que se devuelva y *esPrimo* una variable booleana que indica si el número es primo o no.

```
        double b = Double.valueOf(a).doubleValue();
        int num = 2;
        String resultado;
        boolean esPrimo = true;
```

Para saber si un número dado es primo o no, el algoritmo a seguir es comprobar si es divisible entre algún número comprendido entre 2 y la raíz cuadrada de dicho número.

El bucle *while* del método comprueba el resto para los valores anteriormente mencionados, y cuando encuentra un divisor cambia la variable *esPrimo*.

```
        do
        {
            if (b % num == 0)
                esPrimo = false;
            num++;
        } while (num <= java.lang.Math.sqrt(b));
```

El algoritmo anterior, sin embargo, tiene un punto débil. Si el número elegido es el 2, el resultado sería negativo. Por eso se añade un *if* que contemple esta posibilidad.

```
       if (b==2)
            esPrimo=true;
```

Por último, según el resultado se devuelve un mensaje afirmativo o negativo.

```
        if (esPrimo)
            resultado = "El numero es primo";
        else
            resultado = "El numero no es primo";
        return resultado;
    }
}
```

6.1.1.2 WSDD de JSR-172

Aunque el código del servidor es el mismo para JSR-172 y kSOAP, las particularidades de cada uno hacen necesario un despliegue distinto, por lo que sus archivos WSDL y el formato de los mensajes SOAP serán diferentes en ambos casos.

El archivo WSDD para JSR-172 comienza con una etiqueta *deployment*, con los atributos de espacio de nombres (*xmlns*) y espacio de nombres Java (*xmlns:java*)

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

Abrimos ahora un nuevo elemento *service*, que definirá el servicio web que se está desplegando. Sus atributos son el nombre del servicio, el proveedor y el estilo/uso.

Para el caso de JSR-172 el estilo debe ser *document* o *wrapped*, y el uso *literal*.

```
<service
 name="Math"
 provider="java:RPC"
 style="wrapped"
 use="literal">
```

Definimos ahora dos parámetros; el espacio de nombres de WSDL y el nombre de la clase, que debe incluir el paquete al que pertenece.

```
<parameter
 name="wsdlTargetNamespace"
 value="http://math.samples/"/>

<parameter name="className"
 value="samples.math.Primo"/>
```

La etiqueta operación se refiere al método en que consiste el servicio. Indicamos el nombre de la operación y su nombre cualificado, el espacio de nombres de la operación y del retorno, el tipo de la respuesta y su nombre cualificado.

```
<operation
 name="esPrimo"
 qname="operNS:EsPrimo"
 xmlns:operNS="http://math.samples/"
 returnQName="retNS:PrimoResult"
 xmlns:retNS="http://math.samples/"
 returnType="rtns:string"
 xmlns:rtns="http://www.w3.org/2001/XMLSchema" >
```

Dentro del elemento operación se definen los parámetros que se le pasan a la función con su nombre cualificado, su espacio de nombres y el tipo.

```
<parameter
 qname="pns:A"
 xmlns:pns="http://math.samples/"
 type="tns:string"
 xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
```

Finalmente cerramos todas las etiquetas.

```
      </operation>
  <parameter name="allowedMethods" value="esPrimo"/>
  </service>
</deployment>
```

6.1.1.3 WSDD de kSOAP

El archivo WSDD para kSOAP es mucho más simple que el de JSR-172; la razón es que kSOAP es compatible con los valores por defecto que usa Axis, mientras que JSR-172 no.

Lo primero siempre es la etiqueta *deployment*, cuyos atributos son los mismos que los de JSR-172.

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

En el elemento servicio, los atributos son tan sólo el nombre y el proveedor.

```
<service
 name="Mathks"
 provider="java:RPC">
```

Los parámetros del servicio son el espacio de nombres de WSDL, el nombre de la clase incluyendo el paquete al que pertenece y los métodos permitidos.

```
<parameter
 name="wsdlTargetNamespace"
 value="http://math.samples/"/>

<parameter
 name="className"
 value="samples.mathks.Primo"/>

<parameter
 name="allowedMethods"
 value="esPrimo"/>
```

Por último cerramos las etiquetas.

```
  </service>
</deployment>
```

En éste caso, contrariamente al anterior, no existe el elemento *operation* en el que se definían los parámetros; Axis tomará ésta información al cargar la clase.

6.1.1.4 WSDL de JSR-172

El archivo WSDL de JSR-172 comienza con la definición del espacio de nombres y con un comentario sobre su creación.

```
<wsdl:definitions targetNamespace="http://math.samples/">
<!--
   WSDL created by Apache Axis version: 1.2RC2
   Built on Nov 16, 2004 (12:19:44 EST)
-->
```

Dentro de *types* se encuentra un esquema que contendrá la descripción de los distintos elementos.

```
<wsdl:types>
   <schema
    elementFormDefault="qualified"
    targetNamespace="http://math.samples/">
```

El primer elemento es *esPrimo*, y representa la petición que el cliente realiza al servidor. Consiste en un único elemento, *A*, que será el número entero.

```
<element
 name="esPrimo">
   <complexType>
      <sequence>

         <element
          name="A"
          type="xsd:string"/>
```

```
      </sequence>
   </complexType>
</element>
```

El segundo y último elemento es *EsPrimoResponse*, y es la respuesta que el servidor enviará al cliente. Consiste en una cadena de caracteres de nombre *PrimoResult*, que contendrá el mensaje afirmativo o negativo. Tras ello, cierra las etiquetas del esquema y de *type*.

```
      <element
       name="EsPrimoResponse">
         <complexType>

            <sequence>
               <element
                name="PrimoResult"
                type="xsd:string"/>
            </sequence>

         </complexType>
      </element>
   </schema>
</wsdl:types>
```

A continuación se definen dos mensajes, uno el de petición llamado *EsPrimoRequest* y que está asociado al elemento *EsPrimo* previamente definido, y el otro *EsPrimoResponse* asociado al elemento homónimo.

```
<wsdl:message
 name="EsPrimoResponse">

   <wsdl:part
    element="impl:EsPrimoResponse"
    name="parameters"/>
</wsdl:message>

<wsdl:message
 name="EsPrimoRequest">
   <wsdl:part
    element="impl:EsPrimo"
    name="parameters"/>
</wsdl:message>
```

Se define el servicio de nombre *Primo* a partir de los elementos anteriores: la operación *EsPrimo* recibe el mensaje *EsPrimoRequest* y devuelve el mensaje *EsPrimoResponse*.

```
<wsdl:portType
 name="Primo">
    <wsdl:operation
     name="EsPrimo">

        <wsdl:input
         message="impl:EsPrimoRequest"
         name="EsPrimoRequest"/>

        <wsdl:output
         message="impl:EsPrimoResponse"
         name="EsPrimoResponse"/>
```

```
    </wsdl:operation>
</wsdl:portType>
```

En la etiqueta *binding* definimos las particularidades del servicio. El estilo es *document* y el uso es *literal*, como se especificó en el WSDD.

```
<wsdl:binding
 name="MathSoapBinding"
 type="impl:Primo">

   <wsdlsoap:binding
    style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>

   <wsdl:operation name="EsPrimo">
      <wsdlsoap:operation
       soapAction=""/>

      <wsdl:input
       name="EsPrimoRequest">
         <wsdlsoap:body
          use="literal"/>
      </wsdl:input>

      <wsdl:output
       name="EsPrimoResponse">
         <wsdlsoap:body
          use="literal"/>
      </wsdl:output>

   </wsdl:operation>
</wsdl:binding>
```

Por último especificamos la dirección del servicio.

```
   <wsdl:service
    name="PrimoService">

        <wsdl:port
         binding="impl:MathSoapBinding"
         name="Math">
            <wsdlsoap:address
             location="http://localhost:8080/axis/services/Math"/>
        </wsdl:port>

   </wsdl:service>
</wsdl:definitions>
```

6.1.1.5 WSDL de kSOAP

El archivo WSDL de kSOAP comienza igual que el anterior, con la definición del espacio de nombres y con un comentario sobre su creación.

```
<wsdl:definitions targetNamespace="http://math.samples/">
<!--
   WSDL created by Apache Axis version: 1.2RC2
   Built on Nov 16, 2004 (12:19:44 EST)
-->
```

Definimos dos mensajes: el de petición llamado *esPrimoRequest* que contiene el parámetro de la petición, y el de respuesta llamado *esPrimoResponse* y que contiene el parámetro de la respuesta *esPrimoReturn*.

```
<wsdl:message
 name="esPrimoRequest">

   <wsdl:part
    name="in0"
    type="soapenc:string"/>

</wsdl:message>

<wsdl:message
 name="esPrimoResponse">

   <wsdl:part
    name="esPrimoReturn"
    type="soapenc:string"/>

</wsdl:message>
```

Se define el servicio *Primo*, con la operación *esPrimo* y los mensajes *esPrimoRequest* de entrada y *esPrimoResponse* de salida.

```
<wsdl:portType
 name="Primo">

      <wsdl:operation
       name="esPrimo"
       parameterOrder="in0">

      <wsdl:input
       message="impl:esPrimoRequest"
       name="esPrimoRequest"/>

      <wsdl:output
       message="impl:esPrimoResponse"
       name="esPrimoResponse"/>

   </wsdl:operation>
</wsdl:portType>
```

Definimos ahora las particularidades del servicio; el estilo es *rpc* y el uso es *encoded*, al contrario que en el caso anterior.

```
<wsdl:binding
 name="MathksSoapBinding"
 type="impl:Primo">
   <wsdlsoap:binding
    style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation
    name="esPrimo">
      <wsdlsoap:operation
       soapAction=""/>

         <wsdl:input
          name="esPrimoRequest">
            <wsdlsoap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://mathks.samples"
             use="encoded"/>
```

```
         </wsdl:input>

         <wsdl:output name="esPrimoResponse">
            <wsdlsoap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://math.samples/"
             use="encoded"/>
         </wsdl:output>

   </wsdl:operation>
</wsdl:binding>
```

Por último especificamos la dirección del servicio y cerramos el elemento raíz.

```
   <wsdl:service
    name="PrimoService">

      <wsdl:port
       binding="impl:MathksSoapBinding"
       name="Mathks">
         <wsdlsoap:address
          location="http://localhost:8080/axis/services/Mathks"/>
      </wsdl:port>

   </wsdl:service>
</wsdl:definitions>
```

6.1.2 Cliente

El cliente se encargará de recoger mediante una caja de texto el número proporcionado por el usuario, realizar la llamada al servicio y mostrar por la pantalla un mensaje indicando si el número es primo o no.

Figura 6.1: Pantalla de captura de información

Figura 6.2: Pantalla de resultado

6.1.2.1 Cliente JSR 172

El cliente JSR 172 usado para la prueba del servicio web XML se ha generado automáticamente a partir del archivo WSDL. Lo más interesante es recalcar la facilidad con la que se realiza la llamada al procedimiento remoto.

```
java.lang.String a = inputFields[0].getString();
java.lang.String _returnValue = proxy_Primo_Stub.esPrimo(a);
```

Se observa que todo el proceso es transparente.

6.1.2.2 Cliente kSOAP

Contrariamente al caso del cliente JSR 172, no se dispone de ninguna aplicación que genere automáticamente el cliente kSOAP. Sin embargo, realizando leves modificaciones al cliente anterior dispondremos de otra aplicación muy similar, ideal para realizar la comparación.

La primera modificación es importar las librerías de kSOAP.

```
import org.ksoap.*;
import org.ksoap.transport.*;
import org.ksoap.SoapObject;
```

También será necesario conocer la dirección del servicio web XML.

```
private String soapUrl= "http://localhost:8080/axis/services/Mathks";
```

Por último, la llamada se realiza de una forma distinta y mucho menos transparente. En primer lugar se instancia un objeto SOAP que contendrá nuestro mensaje. Seguidamente se le añade el parámetro que enviaremos, la cadena *a* que contiene el número. Luego abrimos una conexión *http* al servicio y finalmente realizamos la llamada.

```
java.lang.String a = inputFields[0].getString();
SoapObject client = new SoapObject ( "urn:Mathks", "esPrimo");
client.addProperty("a", a);

HttpTransport ht = new HttpTransport(soapUrl,"esPrimo");
java.lang.String _returnValue = ht.call(client).toString();
```

6.1.2.3 Mensajes SOAP JSR 172

Petición

```
<soap:Envelope
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:tns="http://math.samples/">

   <soap:Body>
      <tns:EsPrimo >

         <tns:A>
            77
         </tns:A>

      </tns:EsPrimo>
   </soap:Body>

</soap:Envelope>
```

Respuesta

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

   <soapenv:Body>
      <EsPrimoResponse
       xmlns="http://math.samples/">

         <PrimoResult
          xsi:type="xsd:string">
            El numero es primo
         </PrimoResult>

      </EsPrimoResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

6.1.2.4 Mensajes SOAP kSOAP

Petición

```
<SOAP-ENV:Envelope
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <esPrimo
      xmlns="urn:Mathks"
      id="o0"
      SOAP-ENC:root="1">

         <a
          xmlns=""
          xsi:type="xsd:string">
               77
         </a>

    </esPrimo>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Respuesta

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <ns1:esPrimoResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="urn:Mathks">
         <esPrimoReturn
          xsi:type="soapenc:string"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
           <item
            xsi:type="soapenc:string">
              el numero es primo
           </item>
         </esPrimoReturn>
      </ns1:esPrimoResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

6.1.3 Stub

Las clases conectoras o stub son usadas tan sólo por JSR 172, y como ya hemos visto, son generadas automáticamente por el Wireless Toolkit. La estructura del stub está muy ligada al documento WSDL que lo ha originado.

6.1.3.1 *Primo.java*

El archivo *Primo.java* define la interfaz del método remoto. Es muy simple.

```
package defaultpackage;

public interface Primo extends java.rmi.Remote {
       public java.lang.String esPrimo(java.lang.String a) throws
```

```
java.rmi.RemoteException;

}
```

6.1.3.2 *EsPrimo.java*

EsPrimo.java será un objeto que representa al mensaje de petición. Contiene, en primer lugar, las variables correspondientes a los parámetros del método remoto.

```
package defaultpackage;


public class EsPrimo {
    protected java.lang.String a;
```

Por último, al ser el *string* de tipo *protected*, necesitamos métodos específicos para leer y escribir, *getA* y *setA*.

```
    public EsPrimo() {
    }

    public EsPrimo(java.lang.String a) {
        this.a = a;
    }

    public java.lang.String getA() {
        return a;
    }

    public void setA(java.lang.String a) {
        this.a = a;
    }
}
```

6.1.3.3 *EsPrimoResponse.java*

La clase *EsPrimoResponse* representa al mensaje de respuesta. La estructura es igual que la de *EsPrimo*: el *string* con el resultado, un constructor con parámetros, otro sin ellos y los métodos para establecer y obtener el resultado.

```
package defaultpackage;


public class EsPrimoResponse {
    protected java.lang.String primoResult;

    public EsPrimoResponse() {
    }

    public EsPrimoResponse(java.lang.String primoResult) {
        this.primoResult = primoResult;
    }

    public java.lang.String getPrimoResult() {
        return primoResult;
    }

    public void setPrimoResult(java.lang.String primoResult) {
```

```
        this.primoResult = primoResult;
    }
}
```

6.1.3.4 *Primo_Stub.java*

Primo_Stub es la clase más larga de las anteriores. Es la que se encarga de gestionar toda la comunicación. Comienza importando algunos objetos para el procesado XML.

```
package defaultpackage;

import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.microedition.xml.rpc.Operation;
import javax.microedition.xml.rpc.Type;
import javax.microedition.xml.rpc.ComplexType;
import javax.microedition.xml.rpc.Element;
```

La clase implementará la interfaz *Primo* que ya hemos definido así como *javax.xml.rpc.Stub*

```
public class Primo_Stub implements defaultpackage.Primo, javax.xml.rpc.Stub {
```

Se tienen dos tablas: una de *String*, que almacenará el nombre de las propiedades y la otra de *Object*, que almacenará su valor. Para un índice dado se tiene el par nombre/valor de la propiedad asociada.

```
 private String[] _propertyNames;
 private Object[] _propertyValues;
```

El constructor no recibe ningún parámetro, y lo único que hace es establecer la primera propiedad que será la dirección del servicio.

```
 public Primo_Stub() {
  _propertyNames = new String[] {ENDPOINT_ADDRESS_PROPERTY};
  _propertyValues = new Object[] {"http://localhost:8080/axis/services/Math"};
 }
```

El método *_setProperty* permite añadir una nueva propiedad o variar una ya existente; para ello recibe el nombre que será almacenado en la tabla *_propertyNames* y el valor que se almacenará en *_propertyValues*. Primero recorrerá la tabla para comprobar si la propiedad ya existe. Si existe la modifica, si no aumenta la tabla en un elemento y añade el nuevo par.

```
public void _setProperty(String name, Object value) {
 int size = _propertyNames.length;
 for (int i = 0; i < size; ++i) {
  if (_propertyNames[i].equals(name)) {
   _propertyValues[i] = value;
   return;
   }
  }

// Need to expand our array for a new property

  String[] newPropNames = new String[size + 1];
```

```
  System.arraycopy(_propertyNames, 0, newPropNames, 0, size);
  _propertyNames = newPropNames;
  Object[] newPropValues = new Object[size + 1];
  System.arraycopy(_propertyValues, 0, newPropValues, 0, size);
  _propertyValues = newPropValues;

  _propertyNames[size] = name;
  _propertyValues[size] = value;
 }
```

El siguiente método *_getProperty* permite obtener el valor de una propiedad cuyo nombre se pasa como parámetro. Recorre la tabla buscando dicho nombre y lo devuelve; si no lo encuentra lanza una excepción. Además, el método no permite el acceso a la dirección del servicio, el nombre de usuario o la contraseña.

La propiedad *SESSION_MANTAIN_PROPERTY* indica si se quiere mantener una sesión con el servidor, y la respuesta siempre es negativa.

```
 public Object _getProperty(String name) {
  for (int i = 0; i < _propertyNames.length; ++i) {
   if (_propertyNames[i].equals(name)) {
    return _propertyValues[i];
   }
  }
  if (ENDPOINT_ADDRESS_PROPERTY.equals(name) ||
   USERNAME_PROPERTY.equals(name) ||
   PASSWORD_PROPERTY.equals(name)) {
   return null;
  }
  if (SESSION_MAINTAIN_PROPERTY.equals(name)) {
   return new java.lang.Boolean(false);
  }
  throw new JAXRPCException("Stub does not recognize
                 property: "+name);
 }
```

El método *_prepOperation* permite añadir a una operación dada todas las propiedades almacenadas.

```
 protected void _prepOperation(Operation op) {
  for (int i = 0; i < _propertyNames.length; ++i) {
   op.setProperty(_propertyNames[i],
          _propertyValues[i].toString());
  }
 }
```

El método *esPrimo* será el método local que invoque al servicio web. Se consigue así que la llamada remota parezca local a todos los efectos.

Lo primero que hace es copiar el parámetro en una tabla de objetos; luego crea un objeto *Operation* al que asocia las propiedades almacenadas (que es únicamente la dirección del servicio) y realiza la petición mediante el método *invoke*. Por último se modifican los datos recibidos para adaptarlos al tipo Java que nos interesa, es decir, una cadena de caracteres.

```
public java.lang.String esPrimo(java.lang.String a) throws
java.rmi.RemoteException {
  // Copy the incoming values into an Object array if needed.
  Object[] inputObject = new Object[1];
  inputObject[0] = a;

  Operation op = Operation.newInstance(_qname_EsPrimo,
                _type_EsPrimo, _type_EsPrimoResponse);
  _prepOperation(op);
  op.setProperty(Operation.SOAPACTION_URI_PROPERTY, "");
  Object resultObj;
  try {
   resultObj = op.invoke(inputObject);
  } catch (JAXRPCException e) {
   Throwable cause = e.getLinkedCause();
   if (cause instanceof java.rmi.RemoteException) {
    throw (java.rmi.RemoteException) cause;
   }
   throw e;
  }
  java.lang.String result;
  // Convert the result into the right Java type.
  // Unwrapped return value
  Object primoResultObj = ((Object[])resultObj)[0];
  result = (java.lang.String)primoResultObj;
  return result;
 }
```

Finalmente se definen los nombres cualificados, los elementos usados y los tipos de datos definidos en el documento WSDL usados por el stub.

```
protected static final QName _qname_A =
       new QName("http://math.samples/", "A");
protected static final QName _qname_EsPrimo =
       new QName("http://math.samples/", "EsPrimo");
protected static final QName _qname_EsPrimoResponse =
       new QName("http://math.samples/", "EsPrimoResponse");
protected static final QName _qname_PrimoResult =
       new QName("http://math.samples/", "PrimoResult");

protected static final Element _type_EsPrimo;
protected static final Element _type_EsPrimoResponse;

static {
  // Create all of the Type's that this stub uses, once.
  Element _type_A;
  _type_A = new Element(_qname_A, Type.STRING);
  ComplexType _complexType_esPrimo;
  _complexType_esPrimo = new ComplexType();
  _complexType_esPrimo.elements = new Element[1];
  _complexType_esPrimo.elements[0] = _type_A;
  _type_EsPrimo = new Element(_qname_EsPrimo, _complexType_esPrimo);
  Element _type_PrimoResult;
  _type_PrimoResult = new Element(_qname_PrimoResult, Type.STRING);
  ComplexType _complexType_esPrimoResponse;
  _complexType_esPrimoResponse = new ComplexType();
  _complexType_esPrimoResponse.elements = new Element[1];
  _complexType_esPrimoResponse.elements[0] = _type_PrimoResult;
  _type_EsPrimoResponse = new Element(_qname_EsPrimoResponse,
                            _complexType_esPrimoResponse);
  }
}
```

6.1.4 Comparación

La comparación en consumo de memoria a lo largo del tiempo se puede ver en la siguiente figura:

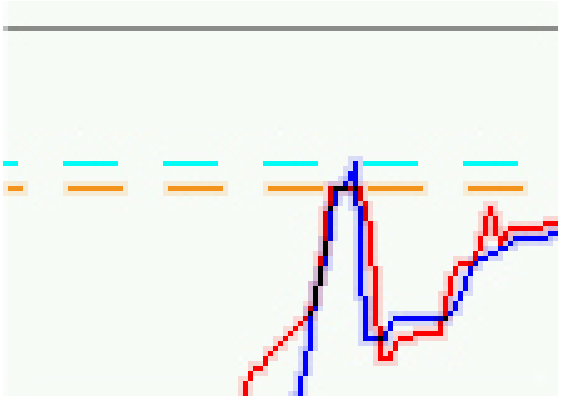

Figura 6.3: Comparación del comportamiento en memoria de kSOAP (rojo) y JSR 172 (azul)

En la siguiente tabla se resumen las características principales de cada una.

	JSR 172	kSOAP
Bytecodes ejecutados	675333	780046
Cambios de hilo	168	121
Clases en el sistema	491	520
Objetos dinámicos	3692	7865
Bytes de objetos dinámicos	130700	351480
Recolecciones de basura	6	27
Bytes recolectados	73556	289584
Tamaño de las librerías	58,5 KB	40,1 KB

Como se puede comprobar, el comportamiento de ambas es bastante similar. Aunque JSR 172 supera a kSOAP en ciertos aspectos, no parece haber un resultado claro sobre cual de las dos es más eficiente.

6.2 Prueba 2: Cartelera de cine

El siguiente servicio es más complejo que el anterior. Consiste en un servicio de cartelera en que el usuario indica el nombre de una película y recibe datos sobre ella como director, actores, argumento y crítica. Es un servicio más realista que el anterior aunque no totalmente viable, ya que los datos de las películas se almacenan en el propio código del servidor. Un servicio real usaría una base de datos para esta tarea; se verá en el siguiente apartado, el servicio de directorio.

Este servicio permitirá comprobar la reacción de JSR 172 y kSOAP frente a mensajes más largos y complejos.

6.2.1 Servidor

El servidor es tan sólo un método desplegado como un servicio web. La creación y la interpretación de los mensajes SOAP es tarea de Axis, siendo totalmente transparente a ésta función.

El servicio es universal y puede ser accedido por cualquier tipo de cliente que conozca las características de la comunicación y que soporte el protocolo SOAP.

6.2.1.1 Servicio

La clase se llama *Cine.java* y contiene el método *peli*, que recibe como parámetro el título de la película de interés en formato *String* y devuelve una tabla de *String* con la información pedida.

```
public class Cine {
    public String[] peli(String a) {
    String[] resultado = new String[9];
```

Lo primero es comprobar el título de la película. Para evitar confusiones se pasa el parámetro recibido a minúsculas antes de la comparación. La única película de la que se disponen datos es “Sin City”.

```
        if (a.toLowerCase().equals("sin city"))
        {
```

Si, efectivamente, se pide información sobre “Sin City” se rellenan los campos de la tabla *resultado* con la información de que disponemos sobre la película.

```
 resultado[0]="SIN CITY";
 resultado[1]="TITULO ORIGINAL: Sin City";
 resultado[2]="DIRECTOR: Frank Miller, Robert Rodriguez";
 resultado[3]="ACTORES: Bruce Willis, Jessica Alba, " +
              "Mickey Rourke, Rosario Dawson";
 resultado[4]="GUIONISTA: Frank Miller, Robert Rodriguez";
 resultado[5]="BASADO EN: Sin City, La Gran Matanza y "+
              "Ese Cobarde Bastardo, de Frank Miller";
 resultado[6]="PUNTUACION: 8/10";
 resultado[7]="ARGUMENTO: Tres historias distintas ambientadas en " +
              "Sin City, con la violencia y la corrupcion como "+
              "denominador comun.";
 resultado[8]="CRITICA: Visualmente demoledora y sadicamente violenta";
}
```

En el caso de que sea una película distinta aquella sobre la que se nos interroga, se devolvería un mensaje de error.

```
        else
        {
            resultado[0]="La película no consta en nuestra base de datos.";
            resultado[1]="Disculpe las molestias.";
            for (int i=2;i<9;i++)
                resultado[i]=".";
        }
```

Finalmente se devuelve el resultado.

```
        return resultado;
    }
}
```

6.2.1.2 WSDD de JSR-172

El archivo WSDD para JSR-172 comienza con una etiqueta *deployment*, con los atributos de espacio de nombres (*xmlns*) y espacio de nombres Java (*xmlns:java*)

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

Abrimos ahora un nuevo elemento *service*, que definirá el servicio web que se está desplegando. Sus atributos son el nombre del servicio, el proveedor y el estilo/uso.

Para el caso de JSR-172 el estilo debe ser *document* o *wrapped*, y el uso *literal*.

```
<service
 name="Cine"
 provider="java:RPC"
 style="wrapped"
 use="literal">
```

Definimos ahora dos parámetros; el espacio de nombres de WSDL y el nombre de la clase, que debe incluir el paquete al que pertenece.

```
<parameter
 name="wsdlTargetNamespace"
 value="http://db.samples/"/>

<parameter name="className"
 value="samples.cine.Cine"/>
```

La etiqueta operación se refiere al método en que consiste el servicio. Indicamos el nombre de la operación y su nombre cualificado, el espacio de nombres de la operación y del retorno, el tipo de la respuesta y su nombre cualificado.

```
<operation
 name="peli"
 qname="operNS:Peli"
 xmlns:operNS="http://db.samples/"
 returnQName="retNS:PeliResult"
 xmlns:retNS="http://db.samples/"
 returnType="rtns:string[]"
 xmlns:rtns="http://www.w3.org/2001/XMLSchema" >
```

Dentro del elemento operación se definen los parámetros que se le pasan a la función con su nombre cualificado, su espacio de nombres y el tipo.

```
<parameter
 qname="pns:datos"
 xmlns:pns="http://db.samples/"
```

```
 type="tns:string"
 xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
```

Finalmente cerramos todas las etiquetas y permitimos el método *peli*.

```
      </operation>
  <parameter
   name="allowedMethods"
   value="peli"/>
  </service>
</deployment>
```

6.2.1.3 WSDD de kSOAP

Lo primero siempre es la etiqueta *deployment*, cuyos atributos son los mismos que los de JSR-172.

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

En el elemento servicio, los atributos son tan sólo el nombre y el proveedor.

```
<service
 name="Cineks"
 provider="java:RPC">
```

Los parámetros del servicio son el espacio de nombres de WSDL, el nombre de la clase incluyendo el paquete al que pertenece y los métodos permitidos.

```
<parameter
 name="wsdlTargetNamespace"
 value="http://db.samples/"/>

<parameter
 name="className"
 value="samples.cineks.Cine"/>

<parameter
 name="allowedMethods"
 value="peli"/>
```

Por último cerramos las etiquetas.

```
  </service>
</deployment>
```

En éste caso, contrariamente al anterior, no existe el elemento *operation* en el que se definían los parámetros; Axis tomará ésta información al cargar la clase correspondiente al servicio.

6.2.1.4 WSDL de JSR-172

El archivo WSDL de JSR-172 comienza con la definición del espacio de nombres y con un comentario sobre su creación.

```
<wsdl:definitions targetNamespace="http://db.samples/">
<!--
   WSDL created by Apache Axis version: 1.2RC2
   Built on Nov 16, 2004 (12:19:44 EST)
-->
```

Dentro de *types* se encuentra un esquema que contendrá la descripción de los distintos elementos.

```
<wsdl:types>
   <schema
    elementFormDefault="qualified"
    targetNamespace="http://db.samples/">
```

El primer elemento es *Peli*, y representa la petición que el cliente realiza al servidor. Consta de una cadena de caracteres con el título de la película, llamada *datos*

```
<element
 name="Peli">
   <complexType>
      <sequence>

         <element
          name="datos"
          type="xsd:string"/>

      </sequence>
   </complexType>
</element>
```

El segundo y último elemento es *PeliResponse*, y es la respuesta que el servidor enviará al cliente. Consiste en un elemento secuencia ilimitado de tipo *string*, es decir, una tabla sin tamaño definido. Tras ello, cierra las etiquetas del esquema y de *type*.

```
      <element
       name="PeliResponse">
         <complexType>

            <sequence>
               <element
                maxOccurs="unbounded"
                name="PeliResult"
                type="xsd:string"/>
            </sequence>

         </complexType>
      </element>
   </schema>
</wsdl:types>
```

A continuación se definen dos mensajes, uno el de petición llamado *PeliRequest* y que está asociado al elemento *Peli* previamente definido, y el otro *PeliResponse* asociado al elemento homónimo.

```
<wsdl:message
 name="PeliResponse">

   <wsdl:part
    element="impl:PeliResponse"
```

```
    name="parameters"/>
</wsdl:message>

<wsdl:message
 name="PeliRequest">
   <wsdl:part
    element="impl:Peli"
    name="parameters"/>
</wsdl:message>
```

Se define el servicio de nombre *Cine* a partir de los elementos anteriores: la operación *Peli* recibe el mensaje *PeliRequest* y devuelve el mensaje *PeliResponse*.

```
<wsdl:portType
 name="Cine">
    <wsdl:operation
     name="Peli">

        <wsdl:input
         message="impl:PeliRequest"
         name="PeliRequest"/>

        <wsdl:output
         message="impl:PeliResponse"
         name="PeliResponse"/>

    </wsdl:operation>
</wsdl:portType>
```

En la etiqueta *binding* definimos las particularidades del servicio. El estilo es *document* y el uso es *literal*, como se especificó en el WSDD.

```
<wsdl:binding
 name="CineSoapBinding"
 type="impl:Cine">

   <wsdlsoap:binding
    style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>

   <wsdl:operation name="Peli">
      <wsdlsoap:operation
       soapAction=""/>

      <wsdl:input
       name="PeliRequest">
         <wsdlsoap:body
          use="literal"/>
      </wsdl:input>

      <wsdl:output
       name="PeliResponse">
         <wsdlsoap:body
          use="literal"/>
      </wsdl:output>

   </wsdl:operation>
</wsdl:binding>
```

Por último especificamos la dirección del servicio.

```
   <wsdl:service
    name="CineService">
```

```
        <wsdl:port
         binding="impl:CineSoapBinding"
         name="Cine">
            <wsdlsoap:address
             location="http://localhost:8080/axis/services/Cine"/>
        </wsdl:port>

   </wsdl:service>
</wsdl:definitions>
```

6.2.1.5 WSDL de kSOAP

El archivo WSDL de kSOAP comienza igual que el anterior, con la definición del espacio de nombres y con un comentario sobre su creación.

```
<wsdl:definitions targetNamespace="http://db.samples/">
<!--
   WSDL created by Apache Axis version: 1.2RC2
   Built on Nov 16, 2004 (12:19:44 EST)
-->
```

En los tipos se incluye un esquema que define un nuevo tipo complejo llamado *ArrayOf_soapenc_string*, que define una tabla de *string*.

```
</wsdl:types>
   <schema
    targetNamespace="http://db.samples/">
         <import
          namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
         <complexType
          name="ArrayOf_soapenc_string">
            <complexContent>

                <restriction
                 base="soapenc:Array">
                       <attribute
                        ref="soapenc:arrayType"
                        wsdl:arrayType="soapenc:string[]"/>

                </restriction>
            </complexContent>
         </complexType>
   </schema>
</wsdl:types>
```

Definimos dos mensajes: el de petición llamado *peliRequest* que contiene los parámetros ya comentados anteriormente, y el de respuesta llamado *peliResponse* y que contiene el tipo complejo definido en el elemento *types*

```
<wsdl:message
 name="peliRequest">

   <wsdl:part
    name="in0"
    type="soapenc:string"/>

</wsdl:message>

<wsdl:message
```

```
 name="peliResponse">

   <wsdl:part
    name="peliReturn"
    type="impl:ArrayOf_soapenc_string"/>

</wsdl:message>
```

Se define el servicio *Cine*, con la operación *peli* y los mensajes *peliRequest* de entrada y *peliResponse* de salida.

```
<wsdl:portType
 name="Cine">

      <wsdl:operation
       name="peli"
       parameterOrder="in0">

      <wsdl:input
       message="impl:peliRequest"
       name="peliRequest"/>

      <wsdl:output
       message="impl:peliResponse"
       name="peliResponse"/>

   </wsdl:operation>
</wsdl:portType>
```

Definimos ahora las particularidades del servicio; el estilo es *rpc* y el uso es *encoded*, al contrario que en el caso anterior.

```
<wsdl:binding
 name="CineksSoapBinding"
 type="impl:Cine">
   <wsdlsoap:binding
    style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation
    name="peli">
      <wsdlsoap:operation
       soapAction=""/>

         <wsdl:input
          name="peliRequest">
            <wsdlsoap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://cineks.samples"
             use="encoded"/>
         </wsdl:input>

         <wsdl:output name="peliResponse">
            <wsdlsoap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://db.samples/"
             use="encoded"/>
         </wsdl:output>

   </wsdl:operation>
</wsdl:binding>
```

Por último especificamos la dirección del servicio y cerramos el elemento raíz.

```
   <wsdl:service
    name="CineService">

      <wsdl:port
       binding="impl:CineksSoapBinding"
       name="Cineks">
         <wsdlsoap:address
          location="http://localhost:8080/axis/services/Cineks"/>
      </wsdl:port>

   </wsdl:service>
</wsdl:definitions>
```

6.2.2 Cliente

El cliente de este servicio lee mediante una caja de texto el título de la película de interés, realiza la petición al servicio web XML y muestra por pantalla la respuesta, formada por los distintos campos de información sobre la película como director, actores, etc.

Figura 6.4: Pantalla de captura de información

Figura 6.5: Pantalla de resultados

6.2.2.1 Cliente JSR 172

El cliente JSR 172 usado para la prueba del servicio web XML se ha generado automáticamente a partir del archivo WSDL. De nuevo, se hace hincapié en la facilidad con la que se realiza la llamada al procedimiento remoto.

```
java.lang.String datos = inputFields[0].getString();
java.lang.String[] _returnValue = proxy_Cine_Stub.peli(datos);
```

Se observa que todo el proceso es transparente.

6.2.2.2 Cliente kSOAP

Contrariamente al caso del cliente JSR 172, no se dispone de ninguna aplicación que genere automáticamente el cliente kSOAP. Sin embargo, realizando leves modificaciones al cliente anterior dispondremos de otra aplicación muy similar, ideal para realizar la comparación.

La primera modificación es importar las librerías de kSOAP.

```
import org.ksoap.*;
import org.ksoap.transport.*;
import org.ksoap.SoapObject;
```

También será necesario conocer la dirección del servicio web XML.

```
private String soapUrl= "http://localhost:8080/axis/services/Cineks";
```

Por último, la llamada se realiza de una forma distinta y mucho menos transparente. En primer lugar se instancia un objeto SOAP que contendrá nuestro mensaje. Seguidamente se le añade el parámetro que enviaremos, la cadena *datos* que contiene el número. Luego abrimos una conexión *http* al servicio y realizamos la llamada.

Por último, se recorre el vector mediante un bucle para convertirlo en una tabla de *String.*

```
java.lang.String datos = inputFields[0].getString();

SoapObject client = new SoapObject ( "urn:Cineks", "peli");
client.addProperty("datos", datos);
HttpTransport ht = new HttpTransport(soapUrl,"peli");
String[] _returnValue = new String[9];
Vector resObj = new Vector();
resObj =(Vector)ht.call(client);

for (int j=0;j<resObj.size();j++)
{
  _returnValue [j] = resObj.elementAt(j).toString();
}
```

6.2.2.3 Mensajes SOAP JSR 172

Petición

```
<soap:Envelope
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:tns="http://db.samples/">

   <soap:Body>
      <tns:Peli>

         <tns:datos>
            sin city
         </tns:datos>

      </tns:Peli>
   </soap:Body>

</soap:Envelope>
```

Respuesta

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

   <soapenv:Body>
      <PeliResponse
       xmlns="http://db.samples/">

         <Result
          xsi:type="xsd:string">
            SIN CITY
         </Result>
```

```
          <Result
           xsi:type="xsd:string">
             TITULO ORIGINAL: Sin City
          </Result>

          <Result
           xsi:type="xsd:string">
             DIRECTOR: Frank Miller, Robert Rodriguez
          </Result>

          <Result
           xsi:type="xsd:string">
            ACTORES: Bruce Willis, Jessica Alba,
            Mickey Rourke, Rosario Dawson
          </Result>

          <Result
           xsi:type="xsd:string">
             GUIONISTA: Frank Miller, Robert Rodriguez
          </Result>

          <Result
           xsi:type="xsd:string">
            BASADO EN: Sin City, La Gran Matanza y
            Ese Cobarde Bastardo, de Frank Miller
          </Result>

          <Result
           xsi:type="xsd:string">
             PUNTUACION: 8/10
          </Result>

          <Result
           xsi:type="xsd:string">
             ARGUMENTO: Tres historias distintas ambientadas en
             Sin City, con la violencia y la corrupcion como
             denominador comun.
          </Result>

          <Result
           xsi:type="xsd:string">
             CRITICA: Visualmente demoledora y sadicamente violenta.
          </Result>


      </PeliResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

6.2.2.4 Mensajes SOAP kSOAP

Petición

```
<SOAP-ENV:Envelope
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <peli
       xmlns="urn:Cineks"
       id="o0"
       SOAP-ENC:root="1">
```

```
          <A
           xmlns=""
           xsi:type="xsd:string">
                sin city
          </A>

    </peli>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Respuesta

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <ns1:peliResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="urn:Cine">
         <peliReturn
          soapenc:arrayType="soapenc:string[8]"
          xsi:type="soapenc:Array"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">

           <item
            xsi:type="soapenc:string">
              SIN CITY
           </item>

           <item
            xsi:type="soapenc:string">
              TITULO ORIGINAL: Sin City
           </item>

           <item
            xsi:type="soapenc:string">
              DIRECTOR: Frank Miller, Robert Rodriguez
           </item>

           <item
            xsi:type="soapenc:string">
              ACTORES: Bruce Willis, Jessica Alba,
              Mickey Rourke, Rosario Dawson
           </item>

           <item
            xsi:type="soapenc:string">
              GUIONISTA: Frank Miller, Robert Rodriguez
           </item>

           <item
            xsi:type="soapenc:string">
              BASADO EN: Sin City, La Gran Matanza y
              Ese Cobarde Bastardo, de Frank Miller
           </item>

           <item
            xsi:type="soapenc:string">
              PUNTUACION: 8/10
           </item>

           <item
            xsi:type="soapenc:string">
              ARGUMENTO: Tres historias distintas ambientadas en
              Sin City, con la violencia y la corrupcion como
              denominador comun.
```

```
            </item>

            <item
             xsi:type="soapenc:string">
               CRITICA: Visualmente demoledora y sadicamente violenta
            </item>

         </peliReturn>
      </ns1:peliResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

6.2.3 Stub

Las clases conectoras o stub son usadas tan sólo por JSR 172, y como ya hemos visto, son generadas automáticamente por el Wireless Toolkit. La estructura del stub está muy ligada al documento WSDL que lo ha originado.

6.2.3.1 *Cine.java*

El archivo *Cine.java* define la interfaz del método remoto. Es muy simple.

```
package defaultpackage;

public interface Cine extends java.rmi.Remote {
       public java.lang.String[] peli(java.lang.String datos) throws
java.rmi.RemoteException;
}
```

6.2.3.2 *Peli.java*

Peli.java será un objeto que representa al mensaje de petición. Contiene, en primer lugar, la variable *datos*, correspondiente al parámetro del método remoto.

```
package defaultpackage;

public class Peli {
    protected java.lang.String datos;
```

Por último, al ser el *string* de tipo *protected*, necesitamos métodos específicos para leer y escribir cada uno. Los métodos serán de la forma *getX* y *setX*.

```
    public Peli() {
    }

    public Peli(java.lang.String datos) {
        this.datos = datos;
    }

    public java.lang.String getDatos() {
        return datos;
    }
```

```
    public void setDatos(java.lang.String datos) {
        this.datos = datos;
    }
}
```

6.2.3.3 *PeliResponse.java*

La clase *PeliResponse* representa al mensaje de respuesta. La estructura es igual que la de *Peli*: la tabla de *string* con el resultado, un constructor con parámetros, otro sin ellos y los métodos para establecer y obtener la tabla.

```
package defaultpackage;


public class PeliResponse {
    protected java.lang.String[] primoResult;

    public PeliResponse() {
    }

    public PeliResponse(java.lang.String[] primoResult) {
        this.primoResult = primoResult;
    }

    public java.lang.String[] getPrimoResult() {
        return primoResult;
    }

    public void setPrimoResult(java.lang.String[] primoResult) {
        this.primoResult = primoResult;
    }
}
```

6.2.3.4 *Cine_Stub.java*

Cine_Stub es la clase más larga de las anteriores. Es la que se encarga de gestionar toda la comunicación. Comienza importando algunos objetos para el procesado XML.

```
package defaultpackage;

import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.microedition.xml.rpc.Operation;
import javax.microedition.xml.rpc.Type;
import javax.microedition.xml.rpc.ComplexType;
import javax.microedition.xml.rpc.Element;
```

La clase implementará la interfaz *Cine* que ya hemos definido así como *javax.xml.rpc.Stub*

```
public class Cine_Stub implements defaultpackage.Cine, javax.xml.rpc.Stub {
```

Se tienen dos tablas: una de *String*, que almacenará el nombre de las propiedades y la otra de *Object*, que almacenará su valor. Para un índice dado se tiene el par nombre/valor de la propiedad asociada.

```
	private String[] _propertyNames;
	private Object[] _propertyValues;
```

El constructor no recibe ningún parámetro, y lo único que hace es establecer la primera propiedad que será la dirección del servicio.

```
public Cine_Stub() {
  _propertyNames = new String[] {ENDPOINT_ADDRESS_PROPERTY};
  _propertyValues = new Object[] {"http://localhost:8080/axis/services/Cine"};
}
```

El método *_setProperty* permite añadir una nueva propiedad o variar una ya existente; para ello recibe el nombre que será almacenado en la tabla *_propertyNames* y el valor que se almacenará en *_propertyValues*. Primero recorrerá la tabla para comprobar si la propiedad ya existe. Si existe la modifica, si no aumenta la tabla en un elemento y añade el nuevo par.

```
public void _setProperty(String name, Object value) {
	int size = _propertyNames.length;
	for (int i = 0; i < size; ++i) {
		if (_propertyNames[i].equals(name)) {
			_propertyValues[i] = value;
			return;
		}
	}
	// Need to expand our array for a new property
	String[] newPropNames = new String[size + 1];
	System.arraycopy(_propertyNames, 0, newPropNames, 0, size);
	_propertyNames = newPropNames;
	Object[] newPropValues = new Object[size + 1];
	System.arraycopy(_propertyValues, 0, newPropValues, 0, size);
	_propertyValues = newPropValues;
		_propertyNames[size] = name;
	_propertyValues[size] = value;
}
```

El siguiente método *_getProperty* permite obtener el valor de una propiedad cuyo nombre se pasa como parámetro. Recorre la tabla buscando dicho nombre y lo devuelve; si no lo encuentra lanza una excepción. Además, el método no permite el acceso a la dirección del servicio, el nombre de usuario o la contraseña.

La propiedad *SESSION_MANTAIN_PROPERTY* indica si se quiere mantener una sesión con el servidor, y la respuesta siempre es negativa.

```
public Object _getProperty(String name) {
	for (int i = 0; i < _propertyNames.length; ++i) {
		if (_propertyNames[i].equals(name)) {
			return _propertyValues[i];
		}
	}
	if (ENDPOINT_ADDRESS_PROPERTY.equals(name) ||
		  USERNAME_PROPERTY.equals(name) ||
		  PASSWORD_PROPERTY.equals(name)) {
```

```
            return null;
      }
      if (SESSION_MAINTAIN_PROPERTY.equals(name)) {
            return new java.lang.Boolean(false);
      }
      throw new JAXRPCException("Stub does not recognize
            property: "+name);
}
```

El método *_prepOperation* permite añadir a una operación dada todas las propiedades almacenadas.

```
protected void _prepOperation(Operation op) {
      for (int i = 0; i < _propertyNames.length; ++i) {
            op.setProperty(_propertyNames[i],
                        _propertyValues[i].toString());
      }
}
```

El método *peli* será el método local que invoque al servicio web. Se consigue así que la llamada remota parezca local a todos los efectos.

Lo primero que hace es copiar todos los parámetros en una tabla de objetos; luego crea un objeto *Operation* al que asocia las propiedades almacenadas (que es únicamente la dirección del servicio) y realiza la petición mediante el método *invoke*. Por último se modifican los datos recibidos para adaptarlos al tipo Java que nos interesa, es decir, una tabla de *strings*.

```
public java.lang.String[] peli(java.lang.String datos)
                  throws java.rmi.RemoteException {
      // Copy the incoming values into an Object array if needed.
      Object[] inputObject = new Object[1];
      inputObject[0] = datos;
            Operation op = Operation.newInstance(_qname_Peli,
                        _type_Peli, _type_PeliResponse);
      _prepOperation(op);
      op.setProperty(Operation.SOAPACTION_URI_PROPERTY, "");
      Object resultObj;
      try {
            resultObj = op.invoke(inputObject);
      } catch (JAXRPCException e) {
            Throwable cause = e.getLinkedCause();
            if (cause instanceof java.rmi.RemoteException) {
                  throw (java.rmi.RemoteException) cause;
            }
            throw e;
      }
      java.lang.String[] result;
      // Convert the result into the right Java type.
      // Unwrapped return value
      Object primoResultObj = ((Object[])resultObj)[0];
      result = (java.lang.String[]) primoResultObj;
      return result;
}
```

Finalmente se definen los nombres cualificados, los elementos usados y los tipos de datos definidos en el documento WSDL usados por el stub.

```
protected static final QName _qname_Peli =
            new QName("http://math.samples/", "Peli");
protected static final QName _qname_PeliResponse =
            new QName("http://math.samples/", "PeliResponse");
protected static final QName _qname_PrimoResult =
            new QName("http://math.samples/", "PrimoResult");
protected static final QName _qname_datos =
            new QName("http://math.samples/", "datos");
protected static final Element _type_Peli;
protected static final Element _type_PeliResponse;
static {
       // Create all of the Type's that this stub uses, once.
       Element _type_datos;
       _type_datos = new Element(_qname_datos, Type.STRING);
       ComplexType _complexType_peli;
       _complexType_peli = new ComplexType();
       _complexType_peli.elements = new Element[1];
       _complexType_peli.elements[0] = _type_datos;
       _type_Peli = new Element(_qname_Peli, _complexType_peli);
       Element _type_PrimoResult;
       _type_PrimoResult =
                new Element(_qname_PrimoResult, Type.STRING, 1, -1, false);
       ComplexType _complexType_peliResponse;
       _complexType_peliResponse = new ComplexType();
       _complexType_peliResponse.elements = new Element[1];
       _complexType_peliResponse.elements[0] = _type_PrimoResult;
       _type_PeliResponse =
                new Element(_qname_PeliResponse, _complexType_peliResponse);
}
```

6.2.4 Comparación

Se puede comparar el consumo de memoria a lo largo del tiempo de ambas aproximaciones en la siguiente figura:

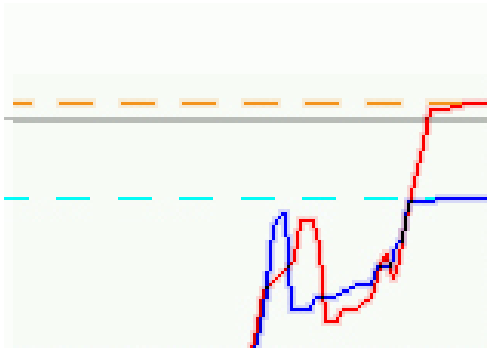

Figura 6.6: Comparación del comportamiento en memoria entre kSOAP (rojo) y JSR 172 (azul)

En la siguiente tabla se resumen las características principales de cada una.

	JSR 172	kSOAP
Bytecodes ejecutados	954880	1144073
Cambios de hilo	149	182
Clases en el sistema	491	520
Objetos dinámicos	4026	10726
Bytes de objetos dinámicos	155856	428988
Recolecciones de basura	6	27

Bytes recolectados	73556	293688
Tamaño de las librerías	58,5 KB	40,1 KB

En este caso la diferencia entre kSOAP y JSR 172 se hace más pronunciada, al consumir kSOAP más memoria que JSR 172 para analizar el mensaje SOAP recibido. Ya se comprueba que JSR 172 va a consumir menos recursos.

6.3 Prueba 3: Servicio de directorio

Esta última prueba pretende ser un ejemplo de servicio real, en que el servidor consulte una base de datos para completar la petición. Consiste en un cliente más complejo que los anteriores que presenta un menú donde se puede elegir una serie de parámetros para realizar una búsqueda en un servicio de páginas amarilllas.

Las funciones básicas son:

- **Recogida de los datos** de la petición SOAP
- Instanciar una **conexión con la base de datos**
- **Montar y enviar la consulta** a la base de datos
- **Almacenar la respuesta** en una tabla y devolverla.

6.3.1 Servidor

El servidor es tan sólo un método remoto desplegado como un servicio web. La creación y la interpretación de los mensajes SOAP es tarea de Axis, siendo totalmente transparente a ésta función.

El servicio es universal y puede ser accedido por cualquier tipo de cliente que conozca las características de la comunicación y que soporte el protocolo SOAP.

6.3.1.1 Servicio

Importamos los paquetes necesarios para procesar la petición SQL y una excepción input/output

```
import java.io.IOException;
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
```

La clase (que se llamará *WKsoap* para kSOAP y *WService* para JSR-172) contiene tan sólo el método *query*, que recibe como parámetros la actividad de interés,

la provincia y la localidad, la categoría en caso de que sea un hotel y el nombre de la empresa en caso de que lo conozcamos.

```
public class WService {
  public String[] query(String Act, String Pro, String Loc, String Cat,
  String Emp) throws Exception {
```

En primer lugar creamos una nueva instancia del driver JDBC que nos proporcionará el acceso a la base de datos.

```
    Class.forName("com.mysql.jdbc.Driver").newInstance();
```

Creamos una conexión con la base de datos MySQL, especificando el nombre de usuario *antonio* y la contraseña vacía.

```
    Connection con = DriverManager.getConnection
          ("jdbc:mysql://localhost:3306/serial", "antonio", "");
```

Ahora tenemos que crear la petición SQL. Para ello usamos los parámetros que hemos recibido: la actividad de interés, la provincia y la localidad, la categoría en el caso de que busquemos un hotel y, de forma opcional, el nombre de la empresa.

```
    String sql ="SELECT DISTINCT * FROM serial WHERE Actividad='" + Act +
        "' and Provincia='" + Pro + "' and Localidad='" + Loc +
        "' and Categoria='" + Cat;
    if (Emp.length()!=0) {
        sql = sql + "' and Empresa='" + Emp;
    }
    sql = sql + "';";
```

Creamos un *statement* para la conexión establecida y le enviamos la petición, guardando el resultado en un *ResultSet*.

```
    Statement stmt = con.createStatement();
    ResultSet record = stmt.executeQuery(sql);
```

Definimos un contador, y calculamos el tamaño del resultado. Para ello vamos al último de los resultados y consultamos el número de la línea en que nos encontramos; ese será el tamaño. Como debemos enviar ocho cadenas de caracteres por cada resultado, multiplicamos el tamaño por ocho. Por último volvemos al principio de los resultados.

```
    int cont=0;
    record.last();
    int Total = 8*record.getRow();
    record.beforeFirst();
```

Instanciamos la tabla de *String* en la que almacenaremos la respuesta. El tamaño es el que se ha calculado anteriormente.

```
    String devolver [] = new String [Total];
```

Ahora entramos en un bucle en el que, por cada resultado de la petición SQL almacenamos en la tabla las ocho características que nos interesan.

```
    while (record.next()) {
        devolver [cont++]=record.getString ("Actividad");
        devolver [cont++]=record.getString ("Provincia");
        devolver [cont++]=record.getString ("Localidad");
        devolver [cont++]=record.getString ("Categoria");
        devolver [cont++]=record.getString ("Empresa");

        devolver [cont++]=record.getString ("Telefono");
        devolver [cont++]=record.getString ("Direccion");
        devolver [cont++]=record.getString ("Referencia");
      }
```

Por último, devolvemos la tabla de cadenas de caracteres.

```
    return devolver;
}
```

6.3.1.2 WSDD de JSR-172

El archivo WSDD para JSR-172 comienza con una etiqueta *deployment*, con los atributos de espacio de nombres (*xmlns*) y espacio de nombres Java (*xmlns:java*)

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

Abrimos ahora un nuevo elemento *service*, que definirá el servicio web que se está desplegando. Sus atributos son el nombre del servicio, el proveedor y el estilo/uso.

Para el caso de JSR-172 el estilo debe ser *document* o *wrapped*, y el uso *literal*.

```
<service
 name="WService"
 provider="java:RPC"
 style="wrapped"
 use="literal">
```

Definimos ahora dos parámetros; el espacio de nombres de WSDL y el nombre de la clase, que debe incluir el paquete al que pertenece.

```
<parameter
 name="wsdlTargetNamespace"
 value="http://db.samples/"/>

<parameter name="className"
 value="WService.WService"/>
```

La etiqueta operación se refiere al método en que consiste el servicio. Indicamos el nombre de la operación y su nombre cualificado, el espacio de nombres de la operación y del retorno, el tipo de la respuesta y su nombre cualificado.

```
<operation
 name="query"
 qname="operNS:Query"
 xmlns:operNS="http://db.samples/"
```

```
 returnQName="retNS:Result"
 xmlns:retNS="http://db.samples/"
 returnType="rtns:string []"
 xmlns:rtns="http://www.w3.org/2001/XMLSchema" >
```

Dentro del elemento operación se definen los parámetros que se le pasan a la función con su nombre cualificado, su espacio de nombres y el tipo.

```
<parameter
 qname="pns:Act"
 xmlns:pns="http://db.samples/"
 type="tns:string"
 xmlns:tns="http://www.w3.org/2001/XMLSchema"/>

<parameter
 qname="pns:Pro"
 xmlns:pns="http://db.samples/"
 type="tns:string"
 xmlns:tns="http://www.w3.org/2001/XMLSchema"/>

<parameter
 qname="pns:Loc"
 xmlns:pns="http://db.samples/"
 type="tns:string"
 xmlns:tns="http://www.w3.org/2001/XMLSchema"/>

<parameter
 qname="pns:Cat"
 xmlns:pns="http://db.samples/"
 type="tns:string"
 xmlns:tns="http://www.w3.org/2001/XMLSchema"/>

<parameter
 qname="pns:Emp"
 xmlns:pns="http://db.samples/"
 type="tns:string"
 xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
```

Finalmente cerramos todas las etiquetas.

```
      </operation>
  <parameter name="allowedMethods" value="query"/>
  </service>
</deployment>
```

6.3.1.3 WSDD de kSOAP

El archivo WSDD para kSOAP es mucho más simple que el de JSR-172; la razón es que kSOAP es compatible con los valores por defecto que usa Axis, mientras que en JSR-172 no.

Lo primero siempre es la etiqueta *deployment*, cuyos atributos son los mismos que los de JSR-172.

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

En el elemento servicio, los atributos son tan sólo el nombre y el proveedor.

```
<service
 name="WKsoap"
 provider="java:RPC">
```

Los parámetros del servicio son el espacio de nombres de WSDL, el nombre de la clase incluyendo el paquete al que pertenece y los métodos permitidos.

```
<parameter
 name="wsdlTargetNamespace"
 value="http://db.samples/"/>

<parameter
 name="className"
 value="WKsoap.WKsoap"/>

<parameter
 name="allowedMethods"
 value="query"/>
```

Por último cerramos las etiquetas.

```
  </service>
</deployment>
```

En éste caso, contrariamente al anterior, no existe el elemento *operation* en el que se definían los parámetros del método *query*. Axis tomará ésta información del propio método.

6.3.1.4 WSDL de JSR-172

El archivo WSDL de JSR-172 comienza con la definición del espacio de nombres y con un comentario sobre su creación.

```
<wsdl:definitions targetNamespace="http://db.samples/">
<!--
   WSDL created by Apache Axis version: 1.2RC2
   Built on Nov 16, 2004 (12:19:44 EST)
-->
```

Dentro de *types* se encuentra un esquema que contendrá la descripción de los distintos elementos.

```
<wsdl:types>
   <schema
    elementFormDefault="qualified"
    targetNamespace="http://db.samples/">
```

El primer elemento es *Query*, y representa la petición que el cliente realiza al servidor. Consta de una secuencia de cinco cadenas de caracteres: la actividad, la provincia, la localidad, la categoría y el nombre de la empresa.

```
<element
 name="Query">
   <complexType>
      <sequence>
```

```
         <element
          name="Act"
          type="xsd:string"/>

         <element name="Pro"
          type="xsd:string"/>

         <element
          name="Loc"
          type="xsd:string"/>

         <element
          name="Cat"
          type="xsd:string"/>

         <element
          name="Emp"
          type="xsd:string"/>
      </sequence>
   </complexType>
</element>
```

El segundo y último elemento es *QueryResponse*, y es la respuesta que el servidor enviará al cliente. Consiste en un elemento secuencia ilimitado de tipo *string*, es decir, una tabla sin tamaño definido. Tras ello, cierra las etiquetas del esquema y de *type*.

```
      <element
       name="QueryResponse">
         <complexType>

            <sequence>
               <element
                maxOccurs="unbounded"
                name="Result"
                type="xsd:string"/>
            </sequence>

         </complexType>
      </element>
   </schema>
</wsdl:types>
```

A continuación se definen dos mensajes, uno el de petición llamado *QueryRequest* y que está asociado al elemento *Query* previamente definido, y el otro *QueryResponse* asociado al elemento homónimo.

```
<wsdl:message
 name="QueryResponse">

   <wsdl:part
    element="impl:QueryResponse"
    name="parameters"/>
</wsdl:message>

<wsdl:message
 name="QueryRequest">
   <wsdl:part
    element="impl:Query"
    name="parameters"/>
</wsdl:message>
```

Se define el servicio de nombre *WService* a partir de los elementos anteriores: la operación *Query* recibe el mensaje *QueryRequest* y devuelve el mensaje *QueryResponse*.

```
<wsdl:portType
 name="WService">
    <wsdl:operation
     name="Query">

        <wsdl:input
         message="impl:QueryRequest"
         name="QueryRequest"/>

        <wsdl:output
         message="impl:QueryResponse"
         name="QueryResponse"/>

    </wsdl:operation>
</wsdl:portType>
```

En la etiqueta *binding* definimos las particularidades del servicio. El estilo es *document* y el uso es *literal*, como se especificó en el WSDD.

```
<wsdl:binding
 name="WServiceSoapBinding"
 type="impl:WService">

   <wsdlsoap:binding
    style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>

   <wsdl:operation name="Query">
      <wsdlsoap:operation
       soapAction=""/>

      <wsdl:input
       name="QueryRequest">
         <wsdlsoap:body
          use="literal"/>
      </wsdl:input>

      <wsdl:output
       name="QueryResponse">
         <wsdlsoap:body
          use="literal"/>
      </wsdl:output>

   </wsdl:operation>
</wsdl:binding>
```

Por último especificamos la dirección del servicio.

```
   <wsdl:service
    name="WServiceService">

        <wsdl:port
         binding="impl:WServiceSoapBinding"
         name="WService">
            <wsdlsoap:address
             location="http://localhost:8080/axis/services/WService"/>
        </wsdl:port>

   </wsdl:service>
```

```
</wsdl:definitions>
```

6.3.1.5 WSDL de kSOAP

El archivo WSDL de kSOAP comienza igual que el anterior, con la definición del espacio de nombres y con un comentario sobre su creación.

```
<wsdl:definitions targetNamespace="http://db.samples/">
<!--
   WSDL created by Apache Axis version: 1.2RC2
   Built on Nov 16, 2004 (12:19:44 EST)
-->
```

En los tipos se incluye un esquema que define un nuevo tipo complejo llamado *ArrayOf_soapenc_string*, que define una tabla de *string*.

```
</wsdl:types>
   <schema
    targetNamespace="http://db.samples/">
         <import
          namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
         <complexType
          name="ArrayOf_soapenc_string">
            <complexContent>

                <restriction
                 base="soapenc:Array">
                       <attribute
                        ref="soapenc:arrayType"
                        wsdl:arrayType="soapenc:string[]"/>

                </restriction>
            </complexContent>
         </complexType>
   </schema>
</wsdl:types>
```

Definimos dos mensajes: el de petición llamado *queryRequest* que contiene los parámetros ya comentados anteriormente, y el de respuesta llamado *queryResponse* y que contiene el tipo complejo definido en el elemento *types*

```
<wsdl:message
 name="queryRequest">

   <wsdl:part
    name="Act"
    type="soapenc:string"/>

   <wsdl:part
    name="Pro"
    type="soapenc:string"/>

   <wsdl:part
    name="Loc"
    type="soapenc:string"/>

   <wsdl:part
    name="Cat"
    type="soapenc:string"/>
```

```
   <wsdl:part
    name="Emp"
    type="soapenc:string"/>

</wsdl:message>

<wsdl:message
 name="queryResponse">

   <wsdl:part
    name="queryReturn"
    type="impl:ArrayOf_soapenc_string"/>

</wsdl:message>
```

Se define el servicio *WKsoap*, con la operación *query* y los mensajes *queryRequest* de entrada y *queryResponse* de salida.

```
<wsdl:portType
 name="WKsoap">

      <wsdl:operation
       name="query"
       parameterOrder="Act Pro Loc Cat Emp">

      <wsdl:input
       message="impl:queryRequest"
       name="queryRequest"/>

      <wsdl:output
       message="impl:queryResponse"
       name="queryResponse"/>

   </wsdl:operation>
</wsdl:portType>
```

Definimos ahora las particularidades del servicio; el estilo es *rpc* y el uso es *encoded*, al contrario que en el caso anterior.

```
<wsdl:binding
 name="WKsoapSoapBinding"
 type="impl:WKsoap">
   <wsdlsoap:binding
    style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation
    name="query">
      <wsdlsoap:operation
       soapAction=""/>

         <wsdl:input
          name="queryRequest">
            <wsdlsoap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://WKsoap"
             use="encoded"/>
         </wsdl:input>

         <wsdl:output name="queryResponse">
            <wsdlsoap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://db.samples/"
             use="encoded"/>
         </wsdl:output>
```

```
   </wsdl:operation>
</wsdl:binding>
```

Por último especificamos la dirección del servicio y cerramos el elemento raíz.

```
   <wsdl:service
    name="WKsoapService">

      <wsdl:port
       binding="impl:WKsoapSoapBinding"
       name="WKsoap">
         <wsdlsoap:address
          location="http://localhost:8080/axis/services/WKsoap"/>
      </wsdl:port>

   </wsdl:service>
</wsdl:definitions>
```

6.3.2 Cliente

El cliente del servicio directorio que se ha usado para las pruebas está basado en la aplicación diseñada por Antonio Albéndiz. Ya que el análisis de dicho cliente no es el objetivo de éste proyecto se hará un resumen de su contenido; para un análisis más exhaustivo del código consultar el proyecto *Aplicación de la serialización sobre J2ME*.

El servicio utilizado es un directorio tanto de empresas como de usuarios particulares con acceso desde cualquier plataforma compatible con servicios web, en nuestro caso J2ME. El cliente tiene como objetivo la presentación de una interfaz gráfica simple que permita el acceso a la base de datos, realizando peticiones mediante un formulario.

Otra funcionalidad añadida es la de callejero; una vez que el usuario ha seleccionado la empresa de su interés la aplicación muestra un pequeño mapa con su localización exacta.

Las funciones básicas son:

- Recogida de información: mediante una serie de pantallas se muestra un formulario que el usuario debe cumplimentar con los objetivos de su búsqueda.
- Una vez recogida la información se monta la petición SOAP y se envía.
- La respuesta SOAP se recibe y se almacenan los datos recibidos en una serie de objetos.
- Los resultados de la búsqueda se muestran por pantalla.
- Una vez se tienen los resultados se ofrece la posibilidad de recibir un mapa donde se indica la localización exacta del lugar.

6.3.2.1 Funcionamiento del programa

La aplicación es muy sencilla e intuitiva. Cualquier usuario, a pesar de no conocer las tecnologías sobre las que se asienta, puede llegar a usar esta aplicación.

En la pantalla inicial se dispone de la posibilidad de entrar en la aplicación o de salir si se ha producido algún fallo. Una vez dentro de la aplicación aparece una pantalla de presentación del servicio.

Figura 6.7: Pantalla de bienvenida

Seleccionando Entrar llegamos a una primera pantalla en la que se van seleccionando las características del servicio que buscamos. Mediante una serie de formularios se recoge la información necesaria: en la primera pantalla se recoge información sobre el servicio de interés y la provincia en que se desea.

Figura 6.8: Primera pantalla de selección

Una vez rellenado, se continúa a la siguiente pantalla, donde se recoge información de la localidad y del número de estrellas deseado en el caso de los hoteles. Se da también la posibilidad de pasar a la pantalla anterior si se ha rellenado erróneamente algún campo.

Figura 6.9: Segunda pantalla de selección

Si se elige continuar, aparecen dos opciones: + *Detalle*, que permite especificar el nombre de la empresa proveedora del servicio que estamos buscando, y *Conecta*, que realiza la petición al servidor. En el caso de que se seleccione un nombre la probabilidad de que se produzcan resultados disminuye bastante.

Figura 6.10: Pantalla de selección de nombre

Una vez seleccionamos *Conecta* se realiza la conexión y aparece una pantalla informativa.

Figura 6.11:Pantalla de conexión

Cuando el cliente recibe la respuesta del servidor desaparece la pantalla anterior y se muestran los resultados por pantalla.

Figura 6.12: Pantalla de resultados

Si seleccionamos *Ver Mapa*, podemos ver una pequeña imagen con la situación de la empresa seleccionada.

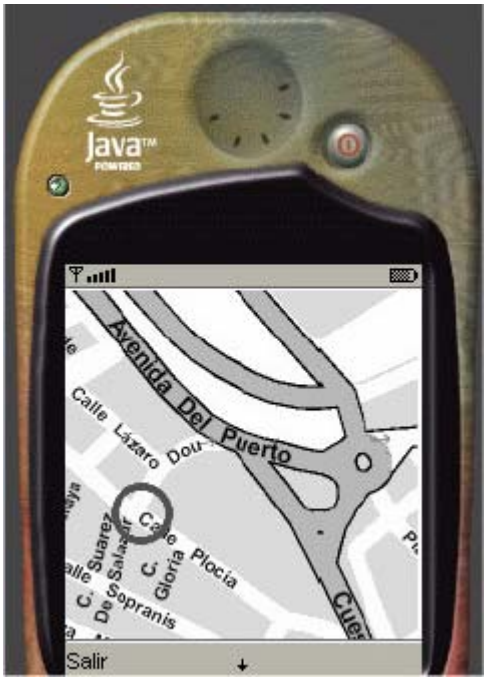

Figura 6.13: Pantalla del mapa

6.3.2.2 Estructura de los datos

La primera decisión a tomar a la hora de proporcionar al cliente el soporte a servicios web es la forma en que se van a enviar los datos. El cliente original serializaba los objetos en una cadena de bytes, así que la primera opción fue esa misma. Sin embargo no era la opción más recomendable, ya que:

- Los mensajes SOAP se envían serializados, así que sería un desperdicio de memoria serializar los mismos datos dos veces.

- Al haber un sólo elemento en la comunicación SOAP se pierde la posibilidad de analizar código XML más complejo

- Al enviar el objeto completo en lugar de los campos necesarios se desperdicia ancho de banda; en el caso de los teléfonos móviles, que suelen tarificar por volumen de datos enviados, esto es poco recomendable.

La opción más favorable fue enviar los datos en cadenas de caracteres. El tamaño de la respuesta se desconoce, ya que no se sabe a priori el número de resultados que va a obtener nuestra búsqueda, por lo que la respuesta será una tabla de cadenas de caracteres.

Si interpretamos el servicio web como una llamada a un procedimiento remoto, el método al que llamaría el cliente se definiría así:

```
String [] query(String Act, String Pro, String Loc, String Cat, String Emp)
```

El método recibe como parámetros la actividad de interés, la provincia y la localidad en que nos encontramos y, de forma opcional, la categoría del hotel y el nombre de la empresa que buscamos. El proceso se detalla más profundamente en el capítulo dedicado al servidor.

6.3.2.3 Cliente JSR 172

Como se ha comentado previamente es innecesario comentar todo el código de la aplicación, ya que es el objetivo de otro proyecto. Sin embargo, sí resulta interesante comentar en profundidad los cambios realizados a la aplicación original.

Lo primero que haremos será crear una instancia del stub; la instanciación se situará en el constructor de la clase *ClienteMid*:

```
  public ClienteMid() {
      display = Display.getDisplay(this);
      proxy_WService_Stub = new Conector.WService_Stub();
    }
```

La aplicación se ejecutará de manera similar hasta que invoque el método que realiza la petición y que recibe la respuesta. Este método se llama *JSR172():*

```
  public void JSR172(){
         try{
```

En primer lugar se instancian algunos objetos y se declaran variables que se usarán posteriormente:

```
boolean varios = false;
String cat = new String();
String emp = new String();
Atributo atrib = new Atributo();
```

Con los datos de los que disponemos completamos el objeto *atrib*, que será el que almacene los datos de una empresa. Algunos datos son opcionales.

```
atrib.setActividad(actividad.getString(actividad.getSelectedIndex()));
atrib.setProvincia(provincias.getString(provincias.getSelectedIndex()));
atrib.setLocalidad(localidades.getString(localidades.getSelectedIndex()));

if (actividad.getSelectedIndex()==7)
{
     atrib.setNombre(auxiliar.getString(auxiliar.getSelectedIndex()));
}

else
{
     if (flag==1)
     {
          atrib.setNombre(empresa.getString());
          emp = atrib.getNombre();
     }

     if(actividad.getSelectedIndex()==1)
          atrib.setCategoria(auxiliar.getSelectedIndex() + 1);
     cat = String.valueOf(atrib.getCategoria());
}
```

Guardamos los datos de la petición en cadenas de caracteres auxiliares.

```
String act = atrib.getActividad();
String prov = atrib.getProvincia();
String loc = atrib.getLocalidad();
```

Para realizar la petición basta con invocar al método *query* del stub. El resultado se almacenará en la tabla *retorno*.

```
String[] retorno = proxy_WService_Stub.query(act,prov,loc,cat,emp);
```

Definimos dos nuevas variables enteras: una es un contador simple con el que iremos recorriendo la tabla de cadenas de caracteres; la otra es un contador decremental que empezará valiendo el número de resultados de la respuesta. Al haber ocho cadenas por cada resultado, el número de resultados se obtiene dividiendo el tamaño de la respuesta entre ocho.

```
int i = 0;
int cont = retorno.length / 8;
```

Se inicia ahora un bucle que irá rellenando el objeto con las cadenas de cada resultado. Tras cada resultado hay que llamar al método Diseña_resultado_parcial, que se encarga de ir acumulando los objetos. También, mediante una bandera, indicamos si el resultado es único o hay varios.

```
while (cont-- > 0)
{
    if (i != 0) {
        Diseña_resultado_parcial(Resultados,nuevoAtributo);
        varios = true;
    }
    atrib.setActividad(retorno[i++]);
    atrib.setProvincia(retorno[i++]);
    atrib.setLocalidad(retorno[i++]);
    atrib.setCategoria(Integer.parseInt(retorno[i++]));
    atrib.setNombre(retorno[i++]);
    atrib.setTelefono(retorno[i++]);
    atrib.setDireccion(retorno[i++]);
    atrib.setReferencia(Integer.parseInt(retorno[i++]));
    atrib.setLocalidad(localidades.getString(localidades.getSelectedIndex()));
}
```

Si hay varios resultados se llama a *Diseña_resultado_parcial* por el último objeto y se llama a un método que lo presenta por pantalla. Si sólo hay un resultado es el método *Diseña_resultado_completo* el que se llama.

```
if (varios)
{
      Diseña_resultado_parcial(Resultados, nuevoAtributo);

      Presenta_resultado_parcial(Resultados);
}
 else
      Diseña_resultado_completo (nuevoAtributo);
```

Por último se tratan las excepciones.

```
catch( Exception e )
{
    System.out.println(e);
    Alert aviso = new Alert("Resultado", "Su búsqueda no ha producido " +
    "resultados. Por favor, pruebe otra vez.",null,null);
    aviso.setTimeout(Alert.FOREVER);
    display.setCurrent(aviso,display.getCurrent());
}
```

6.3.2.4 Cliente kSOAP

Contrariamente a lo que sucedía en JSR-172, con kSOAP sí es necesario importar las librerías en la clase principal.

```
import org.ksoap.*;
import org.ksoap.transport.*;
import org.ksoap.SoapObject;
```

En éste caso la función que realizará las tareas se llama *ksoap*

```
  public void ksoap() {
          try{
```

Las variables, la instanciación de objetos y el rellenado del objeto de petición son idénticos al caso anterior.

```
boolean varios = false;
String cat = new String();
String emp = new String();
Atributo atrib = new Atributo();

atrib.setActividad
          (actividad.getString(actividad.getSelectedIndex()));
atrib.setProvincia
          (provincias.getString(provincias.getSelectedIndex()));
atrib.setLocalidad
          (localidades.getString(localidades.getSelectedIndex()));

if (actividad.getSelectedIndex()==7)
{
    atrib.setNombre(auxiliar.getString(auxiliar.getSelectedIndex()));
}

else
{
    if (flag==1)
    {
        atrib.setNombre(empresa.getString());
        emp = atrib.getNombre();
    }

    if(actividad.getSelectedIndex()==1)
        atrib.setCategoria(auxiliar.getSelectedIndex() + 1);
        cat = String.valueOf(atrib.getCategoria());
}

String act = atrib.getActividad();
String prov = atrib.getProvincia();
String loc = atrib.getLocalidad();
```

En primer lugar instanciamos el objeto SOAP que contendrá la petición. Para ello tenemos que pasar como parámetro el nombre del servicio web y el nombre del método al que llamamos.

```
SoapObject client = new SoapObject ( "urn:WKsoap", "query");
```

Completamos la petición añadiendo los datos (propiedades) del método remoto.

```
client.addProperty("Act", act);
client.addProperty("Pro", prov);
client.addProperty("Loc", loc);
client.addProperty("Cat", cat);
client.addProperty("Emp", emp);
```

Instanciamos un *HttpTransport* y enviamos la petición, almacenando el resultado en el nuevo vector *resobj*

```
HttpTransport ht = new HttpTransport(soapUrl,"query");
Vector resobj = new Vector();
resobj = (Vector)ht.call(client);
```

El resultado es un vector de objetos; lo convertimos en una tabla de cadenas, que es lo que era en un principio.

```
String[] retorno = new String [resobj.size()];
for (int j=0;j<resobj.size();j++){
      retorno [j] = resobj.elementAt(j).toString();
}
```

El resto del código es similar al caso anterior.

```
int i = 0;
int cont = resobj.size()/8;

while (cont-- > 0)
{
     if (i != 0) {
          Diseña_resultado_parcial(Resultados,nuevoAtributo);
          varios = true;
     }
     atrib.setActividad(retorno[i++]);
     atrib.setProvincia(retorno[i++]);
     atrib.setLocalidad(retorno[i++]);
     atrib.setCategoria(Integer.parseInt(retorno[i++]));
     atrib.setNombre(retorno[i++]);
     atrib.setTelefono(retorno[i++]);
     atrib.setDireccion(retorno[i++]);
     atrib.setReferencia(Integer.parseInt(retorno[i++]));
     atrib.setLocalidad
           (localidades.getString(localidades.getSelectedIndex()));
     }
     if (varios)
     {
          Diseña_resultado_parcial(Resultados, atrib);

          Presenta_resultado_parcial(Resultados);
     }
     else
          Diseña_resultado_completo (atrib);
}
catch( Exception e )
{
      System.out.println(e);
      Alert aviso = new Alert("Resultado", "Su búsqueda no ha producido " +
      "resultados. Por favor, pruebe otra vez.",null,null);
      aviso.setTimeout(Alert.FOREVER);
      display.setCurrent(aviso,display.getCurrent());
}
```

6.3.2.5 Mensajes SOAP JSR 172

Petición

```
<soap:Envelope
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:tns="http://db.samples/">

   <soap:Body>
      <tns:Query>

         <tns:Act>
            Autoescuela
```

```
         </tns:Act>

         <tns:Pro>
            Sevilla
         </tns:Pro>

         <tns:Loc>
            Sevilla
         </tns:Loc>

         <tns:Cat>
         </tns:Cat>

         <tns:Emp>
            Leonesa
         </tns:Emp>

      </tns:Query>
   </soap:Body>

</soap:Envelope>
```

Respuesta

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

   <soapenv:Body>
      <QueryResponse
       xmlns="http://db.samples/">

         <Result
          xsi:type="xsd:string">
            Autoescuela
         </Result>

         <Result
          xsi:type="xsd:string">
            Sevilla
         </Result>

         <Result
          xsi:type="xsd:string">
            Sevilla
         </Result>

         <Result
          xsi:type="xsd:string">
            0
         </Result>

         <Result
          xsi:type="xsd:string">
            Leonesa
         </Result>

         <Result
          xsi:type="xsd:string">
            954232506
         </Result>

         <Result
          xsi:type="xsd:string">
```

```
            Reina Mercedes, 1
          </Result>

          <Result
           xsi:type="xsd:string">
             166
          </Result>

       </QueryResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

6.3.2.6 Mensajes SOAP kSOAP

Petición

```
<SOAP-ENV:Envelope
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <query
      xmlns="urn:WKsoap"
      id="o0"
      SOAP-ENC:root="1">

         <Act
          xmlns=""
          xsi:type="xsd:string">
               Autoescuela
         </Act>

         <Pro
          xmlns=""
          xsi:type="xsd:string">
               Sevilla
         </Pro>

         <Loc
          xmlns=""
          xsi:type="xsd:string">
               Sevilla
         </Loc>

         <Cat
          xmlns=""
          xsi:type="xsd:string">
         </Cat>

         <Emp
          xmlns=""
          xsi:type="xsd:string">
              Leonesa
         </Emp>

    </query>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Respuesta

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <ns1:queryResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="urn:WKsoap">
         <queryReturn
          soapenc:arrayType="soapenc:string[8]"
          xsi:type="soapenc:Array"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">

            <item
             xsi:type="soapenc:string">
               Autoescuela
            </item>

            <item
             xsi:type="soapenc:string">
               Sevilla
            </item>

            <item
             xsi:type="soapenc:string">
               Sevilla
            </item>

            <item
             xsi:type="soapenc:string">
               0
            </item>

            <item
             xsi:type="soapenc:string">
               Leonesa
            </item>

            <item
             xsi:type="soapenc:string">
               954232506
            </item>

            <item
             xsi:type="soapenc:string">
               Reina Mercedes, 1
            </item>

            <item
             xsi:type="soapenc:string">
               166
            </item>

         </queryReturn>
      </ns1:queryResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

6.3.3 Stub

Las clases conectoras o stub son usadas tan sólo por JSR 172, y como ya hemos visto, son generadas automáticamente por el Wireless Toolkit. La estructura del stub está muy ligada al documento WSDL que lo ha originado.

6.3.3.1 *WService.java*

El archivo *WService.java* define la interfaz del método remoto. Es muy simple.

```
package Conector;

public interface WService extends java.rmi.Remote {
	public java.lang.String[] query(java.lang.String act,
		java.lang.String pro,
		java.lang.String loc,
		java.lang.String cat,
		java.lang.String emp) throws java.rmi.RemoteException;

}
```

6.3.3.2 *Query.java*

Query.java será un objeto que representa al mensaje de petición. Contiene, en primer lugar, las variables correspondientes a los parámetros del método remoto.

```
package Conector;


public class Query {
    protected java.lang.String act;
    protected java.lang.String pro;
    protected java.lang.String loc;
    protected java.lang.String cat;
    protected java.lang.String emp;
```

Por último, al ser los *string* de tipo *protected*, necesitamos métodos específicos para leer y escribir cada uno. Los métodos serán de la forma *getX* y *setX*.

```
    public java.lang.String getAct() {
        return act;
    }

    public void setAct(java.lang.String act) {
        this.act = act;
    }

    public java.lang.String getPro() {
        return pro;
    }

    public void setPro(java.lang.String pro) {
        this.pro = pro;
    }

    public java.lang.String getLoc() {
        return loc;
    }

    public void setLoc(java.lang.String loc) {
        this.loc = loc;
    }
```

```
    public java.lang.String getCat() {
        return cat;
    }

    public void setCat(java.lang.String cat) {
        this.cat = cat;
    }

    public java.lang.String getEmp() {
        return emp;
    }

    public void setEmp(java.lang.String emp) {
        this.emp = emp;
    }
}
```

6.3.3.3 *QueryResponse.java*

La clase *QueryResponse* representa al mensaje de respuesta. La estructura es igual que la de *Query*: la tabla de *string* con el resultado, un constructor con parámetros, otro sin ellos y los métodos para establecer y obtener la tabla.

```
package Conector;


public class QueryResponse {
    protected java.lang.String[] result;

    public QueryResponse() {
    }

    public QueryResponse(java.lang.String[] result) {
        this.result = result;
    }

    public java.lang.String[] getResult() {
        return result;
    }

    public void setResult(java.lang.String[] result) {
        this.result = result;
    }
}
```

6.3.3.4 *WService_Stub.java*

WService_Stub es la clase más larga de las anteriores. Es la que se encarga de gestionar toda la comunicación. Comienza importando algunos objetos para el procesado XML.

```
package Conector;

import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.microedition.xml.rpc.Operation;
import javax.microedition.xml.rpc.Type;
import javax.microedition.xml.rpc.ComplexType;
```

```
import javax.microedition.xml.rpc.Element;
```

La clase implementará la interfaz *WService* que ya hemos definido así como *javax.xml.rpc.Stub*

```
public class WService_Stub implements
      Conector.WService,
      javax.xml.rpc.Stub {
```

Se tienen dos tablas: una de *String*, que almacenará el nombre de las propiedades y la otra de *Object*, que almacenará su valor. Para un índice dado se tiene el par nombre/valor de la propiedad asociada.

```
      private String[] _propertyNames;
      private Object[] _propertyValues;
```

El constructor no recibe ningún parámetro, y lo único que hace es establecer la primera propiedad que será la dirección del servicio.

```
      public WService_Stub() {
            _propertyNames = new String[] {ENDPOINT_ADDRESS_PROPERTY};
            _propertyValues = new Object[]
                  {"http://localhost:8080/axis/services/WService"};
      }
```

El método *_setProperty* permite añadir una nueva propiedad o variar una ya existente; para ello recibe el nombre que será almacenado en la tabla *_propertyNames* y el valor que se almacenará en *_propertyValues*. Primero recorrerá la tabla para comprobar si la propiedad ya existe. Si existe la modifica, si no aumenta la tabla en un elemento y añade el nuevo par.

```
      public void _setProperty(String name, Object value) {
            int size = _propertyNames.length;
            for (int i = 0; i < size; ++i) {
                  if (_propertyNames[i].equals(name)) {
                        _propertyValues[i] = value;
                        return;
                  }
            }
            // Need to expand our array for a new property
            String[] newPropNames = new String[size + 1];
            System.arraycopy(_propertyNames, 0, newPropNames,
                  0, size);
            _propertyNames = newPropNames;
            Object[] newPropValues = new Object[size + 1];
            System.arraycopy(_propertyValues, 0, newPropValues,
                  0, size);
            _propertyValues = newPropValues;

            _propertyNames[size] = name;
            _propertyValues[size] = value;
      }
```

El siguiente método *_getProperty* permite obtener el valor de una propiedad cuyo nombre se pasa como parámetro. Recorre la tabla buscando dicho nombre y lo

devuelve; si no lo encuentra lanza una excepción. Además, el método no permite el acceso a la dirección del servicio, el nombre de usuario o la contraseña.

La propiedad *SESSION_MANTAIN_PROPERTY* indica si se quiere mantener una sesión con el servidor, y la respuesta siempre es negativa.

```
public Object _getProperty(String name) {
       for (int i = 0; i < _propertyNames.length; ++i) {
              if (_propertyNames[i].equals(name)) {
                     return _propertyValues[i];
              }
       }
       if (ENDPOINT_ADDRESS_PROPERTY.equals(name) ||
          USERNAME_PROPERTY.equals(name) ||
          PASSWORD_PROPERTY.equals(name)) {
              return null;
       }
       if (SESSION_MAINTAIN_PROPERTY.equals(name)) {
              return new java.lang.Boolean(false);
       }
       throw new JAXRPCException("Stub does not recognize
            property: "+name);
}
```

El método *_prepOperation* permite añadir a una operación dada todas las propiedades almacenadas.

```
protected void _prepOperation(Operation op) {
       for (int i = 0; i < _propertyNames.length; ++i) {
              op.setProperty(_propertyNames[i],
                     _propertyValues[i].toString());
       }
}
```

El método *query* será el método local que invoque al servicio web. Se consigue así que la llamada remota parezca local a todos los efectos.

Lo primero que hace es copiar todos los parámetros en una tabla de objetos; luego crea un objeto *Operation* al que asocia las propiedades almacenadas (que es únicamente la dirección del servicio) y realiza la petición mediante el método *invoke*. Por último se modifican los datos recibidos para adaptarlos al tipo Java que nos interesa, es decir, una tabla de *strings*.

```
public java.lang.String[] query(java.lang.String act,
                                java.lang.String pro,
                                java.lang.String loc,
                                java.lang.String cat,
                                java.lang.String emp)
                         throws java.rmi.RemoteException {
       //Copy the incoming values into an Object array if needed.
       Object[] inputObject = new Object[5];
       inputObject[0] = act;
       inputObject[1] = pro;
       inputObject[2] = loc;
       inputObject[3] = cat;
       inputObject[4] = emp;
```

```
            Operation op = Operation.newInstance(_qname_Query, _type_Query,
_type_QueryResponse);
            _prepOperation(op);
            op.setProperty(Operation.SOAPACTION_URI_PROPERTY, "");
            Object resultObj;
            try {
                  resultObj = op.invoke(inputObject);
            } catch (JAXRPCException e) {
                  Throwable cause = e.getLinkedCause();
                  if (cause instanceof java.rmi.RemoteException) {
                        throw (java.rmi.RemoteException) cause;
                  }
                  throw e;
            }
            java.lang.String[] result;
            // Convert the result into the right Java type.
            // Unwrapped return value
            Object resultObj2 = ((Object[])resultObj)[0];
            result = (java.lang.String[]) resultObj2;
            return result;
      }
```

Finalmente se definen los nombres cualificados, los elementos usados y los tipos de datos definidos en el documento WSDL usados por el stub.

```
      protected static final QName _qname_Act =
            new QName("http://db.samples/", "Act");
      protected static final QName _qname_Cat =
            new QName("http://db.samples/", "Cat");
      protected static final QName _qname_Emp =
            new QName("http://db.samples/", "Emp");
      protected static final QName _qname_Loc =
            new QName("http://db.samples/", "Loc");
      protected static final QName _qname_Pro =
            new QName("http://db.samples/", "Pro");
      protected static final QName _qname_Query =
            new QName("http://db.samples/", "Query");
      protected static final QName _qname_QueryResponse =
            new QName("http://db.samples/", "QueryResponse");
      protected static final QName _qname_Result =
            new QName("http://db.samples/", "Result");
      protected static final Element _type_Query;
      protected static final Element _type_QueryResponse;
      static {
            // Create all of the Type's that this stub uses, once.
            Element _type_Act;
            _type_Act = new Element(_qname_Act, Type.STRING);
            Element _type_Pro;
            _type_Pro = new Element(_qname_Pro, Type.STRING);
            Element _type_Loc;
            _type_Loc = new Element(_qname_Loc, Type.STRING);
            Element _type_Cat;
            _type_Cat = new Element(_qname_Cat, Type.STRING);
            Element _type_Emp;
            _type_Emp = new Element(_qname_Emp, Type.STRING);
            ComplexType _complexType_query;
            _complexType_query = new ComplexType();
            _complexType_query.elements = new Element[5];
            _complexType_query.elements[0] = _type_Act;
            _complexType_query.elements[1] = _type_Pro;
            _complexType_query.elements[2] = _type_Loc;
            _complexType_query.elements[3] = _type_Cat;
```

```
            _complexType_query.elements[4] = _type_Emp;
            _type_Query = new Element(_qname_Query,
                        _complexType_query);
            Element _type_Result;
            _type_Result = new Element(_qname_Result, Type.STRING,
                       1, -1, false);
            ComplexType _complexType_queryResponse;
            _complexType_queryResponse = new ComplexType();
            _complexType_queryResponse.elements = new Element[1];
            _complexType_queryResponse.elements[0] = _type_Result;
            _type_QueryResponse = new Element(_qname_QueryResponse,
                        _complexType_queryResponse);
      }

}
```

6.3.4 Comparación

La comparación en memoria a lo largo del tiempo de ambas aproximaciones se muestra en la siguiente figura:

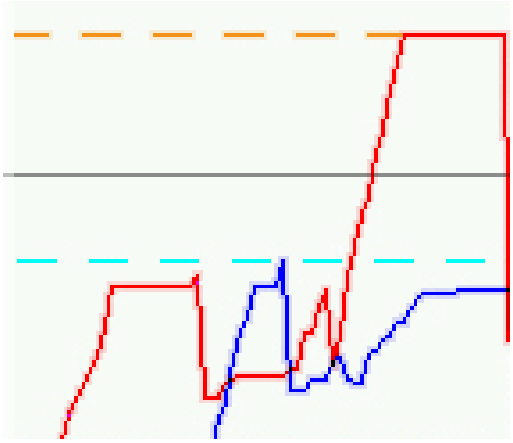

Figura 6.14: Comparación del rendimiento en memoria de kSOAP (rojo) y JSR 172 (azul)

En la siguiente tabla se resumen las características principales de cada una.

	JSR 172	kSOAP
Bytecodes ejecutados	1087350	1363041
Cambios de hilo	149	89
Clases en el sistema	493	522
Objetos dinámicos	4737	14177
Bytes de objetos dinámicos	178756	509104
Recolecciones de basura	11	32
Bytes recolectados	108228	318644
Tamaño de las librerías	58,5 KB	40,1 KB

En este caso la diferencia en memoria ha aumentado hasta más del doble, ya que en este caso el procesado XML es más complejo.

Además, en la tabla anterior se puede comprobar que a pesar de que las librerías de JSR 172 son más voluminosas que las de kSOAP, su rendimiento en ejecución es mucho más ligero con una diferencia abismal.

La conclusión es que el consumo de memoria de kSOAP aumenta proporcionalmente a la complejidad de los mensajes intercambiados, mientras que JSR 172 mantiene un consumo limitado en cualquier caso. Por tanto se deduce que JSR 172 es una opción mucho más adecuada para el acceso a servicios web XML.