```
...
HttpSession sesion=request.getSession(false);
if (sesion!=null) {
      String BOTON=request.getParameter("BOTON");
      Usuario usu=(Usuario) sesion.getAttribute("usuario");
      Broker bd=(Broker) sesion.getAttribute("bd");

      if (BOTON.equals("Guardar nuevo")) {
            try {
                  Cliente o=new Cliente();
                  cargaObjeto(request, o, usu, sesion, IConstantesBotones.SAVE_NEW);
                  o.insert(bd);
                  out.println("<html><body> ….  </body></html>");
                  }
                  catch (Exception e) {
                        sesion.setAttribute("mensaje", e.getMessage());
                        out.println("<html><body> ….  </body></html>");
                  }
            }
      }
      ...
```

Figura 161. Fragmento del método *doPost* en el servlet que recibe los datos del formulario mostrado en la Figura 160

2.04 Ejercicio

El sistema de gestión bancaria permite la manipulación de los datos de su base de datos mediante una aplicación web y una aplicación de escritorio que comparten capa de dominio. Ambas aplicaciones permiten la obtención de listados de, por ejemplo, clientes, como se muestra en la siguiente figura:

Figura 162

Para la obtención del listado de la derecha, el programador añadió un método *getListado* a la clase *dominio.Cliente,* que devuelve en formato HTML la tabla que se muestra. Explique si esta solución es adecuada o inadecuada y por qué.

2.05 Ejercicio

¿Qué desventajas tiene usar un agente de base de datos singleton en una aplicación web?

3. Introducción a los Servicios web

Mediante los servicios web, un cliente puede ejecutar un método en un equipo remoto, transportando la llamada (hacia el servidor) y el resultado (desde el servidor al

cliente) mediante protocolo *http* (normalmente). La idea es servir la misma funcionalidad que permiten otros sistemas de invocación remota de métodos, como RMI (que vimos en el capítulo 7), pero de un modo más portable.

La portabilidad se consigue gracias a que todo el intercambio de información entre cliente y servidor se realiza en SOAP (*Simple Object Access Protocol*), que un protocolo de mensajería basado en XML: así, la llamada a la operación consiste realmente en la transmisión de un mensaje SOAP, el resultado devuelto también, etc. De este modo, el cliente puede estar construido en Java y el servidor en .NET, pero ambos conseguirán comunicarse gracias a la estructura de los mensajes que intercambian.

Los servidores ofrecen una descripción de sus servicios web en WSDL (*Web Services Description Language*), que es una representación en XML del servicio ofrecido. Así, un cliente puede conocer los métodos ofrecidos por el servidor, sus parámetros con sus tipos, etc., simplemente consultando el correspondiente documento WSDL.

3.01 WSDL

Supongamos que nuestro sistema de gestión bancario utiliza, para validar las operaciones realizadas con tarjeta de crédito, el siguiente método remoto:

public boolean validar(String numeroDeTarjeta, double importe)

Si este método es accesible como un servicio web, debe estar descrito en WSDL, por ejemplo, del siguiente modo:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--
Generated by the Oracle9i JDeveloper Web Services WSDL Generator
-->
- <!--
Date Created: Thu Apr 22 11:00:35 CEST 2004
-->
- <definitions name="VisaWS" targetNamespace="http://visa/dominio/Visa.wsdl"
      xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://visa/dominio/Visa.wsdl"
      xmlns:ns1="http://visa.dominio/IVisaWS.xsd">
  - <types>
      <schema targetNamespace="http://visa.dominio/IVisaWS.xsd"
          xmlns="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
          ENC="http://schemas.xmlsoap.org/soap/encoding/" />
    </types>
  - <message name="validar0Request">
      <part name="numeroDeTarjeta" type="xsd:string" />
      <part name="importe" type="xsd:double" />
    </message>
  - <message name="validar0Response">
      <part name="return" type="xsd:boolean" />
    </message>
  - <portType name="VisaPortType">
    - <operation name="validar">
        <input name="validar0Request" message="tns:validar0Request" />
        <output name="validar0Response" message="tns:validar0Response" />
      </operation>
```

```
    </portType>
  - <binding name="VisaBinding" type="tns:VisaPortType">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    - <operation name="validar">
        <soap:operation soapAction="" style="rpc" />
      - <input name="validar0Request">
          <soap:body use="encoded" namespace="VisaWS"
              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </input>
      - <output name="validar0Response">
          <soap:body use="encoded" namespace="VisaWS"
              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </output>
      </operation>
    </binding>
  - <service name="VisaWS">
    - <port name="VisaPort" binding="tns:VisaBinding">
        <soap:address location="" />
      </port>
    </service>
</definitions>
```

Figura 163. Descripción en WSDL del método de validación anterior

De la figura anterior merece la pena destacar algunos elementos:

`- <message name="validar0Request">`
`  <part name="numeroDeTarjeta"`
`  type="xsd:string" />`
`  <part name="importe" type="xsd:double" />`
`</message>`
	Nombre del método accesible de forma remota, nombres y tipos de los parámetros. El postfijo *Request* denota el formato en que debe enviarse la solicitud al servidor. Cuando el cliente invoca el servicio, envía un mensaje *validar0Request.*

`- <message name="validar0Response">`
`  <part name="return" type="xsd:boolean" />`
`</message>`
	Tipo del resultado devuelto por el método. El postfijo *Response* se refiere precisamente a que es el tipo devuelto lo que se está representando.

`- <portType name="VisaPortType">`
`  - <operation name="validar">`
`  <input name="validar0Request"`
`  message="tns:validar0Request" />`
`  <output name="validar0Response"`
`  message="tns:validar0Response" />`
`  </operation>`
`  </portType>`
	Operaciones que conforman la interfaz del servicio *validar*, que se corresponden con los dos *messages* anteriores.

Figura 164. Significado de algunos elementos del WSDL mostrado en la figura anterior

Los entornos de desarrollo recientes incluyen los *add-ins* necesarios para generar la especificación WSDL de una clase.

3.02 Escritura de un cliente que acceda a un servicio web

El cliente que utiliza el servicio web necesita una clase que actúe como proxy entre él mismo y el servicio web ofertado por el servidor. Cuando el proxy recibe del cliente una solicitud de llamada al servicio web, el proxy la traduce a un mensaje SOAP, que envía al servidor; éste, entonces, lo ejecuta, y devuelve un mensaje SOAP al proxy; és-

te, entonces, traduce el mensaje a objetos Java, .NET, etc. y entrega el resultado al cliente que efectuó la petición.

La siguiente figura muestra la relación entre el proxy, el servicio web y la clase que lo implementa: el servicio web (en el centro) ofrece a los clientes acceso a la operación *validar(String, Double);* la operación se encuentra realmente implementada en la clase situada abajo (*Visa*), y podría incluir llamadas a otros métodos de otras clases, acceso a una base de datos, acceso a otros servicios web, etc. El elemento de la izquierda (*VisaWSStub*) es la clase que actúa de proxy entre los clientes y el servicio web. Nótese que esta clase incluye, además de otras, la operación *validar(String, Double)*. El proxy mostrado se ha obtenido de forma automática con un entorno de desarrollo, por lo que sus miembros pueden variar de unos casos a otros. El atributo *_endpoint* representa la URL en la que se encuentra publicado el servicio web.

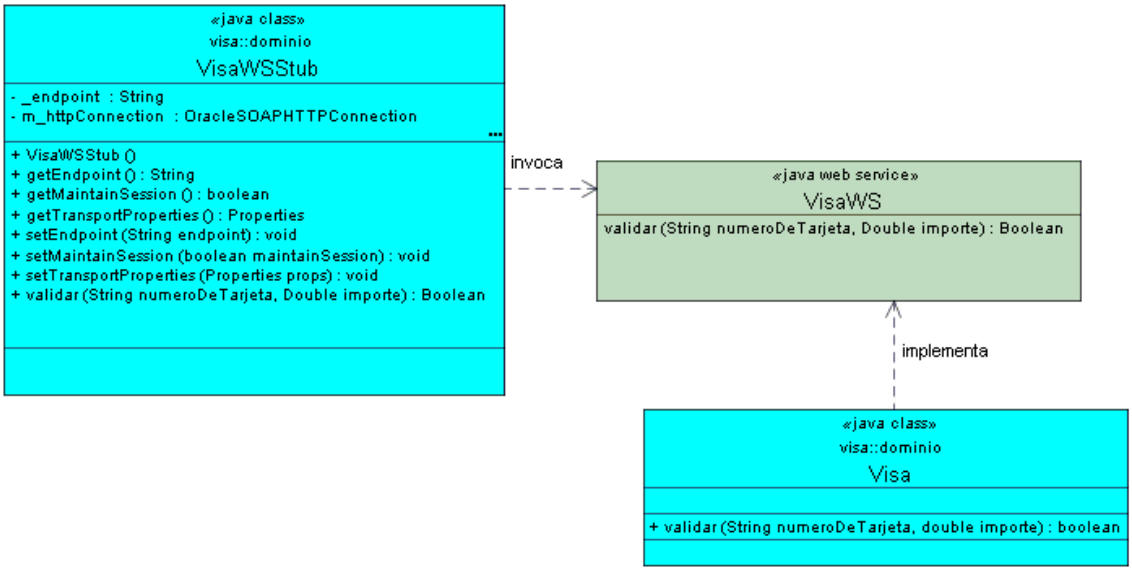

Figura 165. El proxy, el servicio web y la clase que lo implementa

La aplicación cliente hace entonces uso del proxy para acceder al servicio web, por ejemplo con un trozo de código como el que sigue:

```
protected void validarDisponibilidadDeCredito(double importe)
                                                    throws Exception
{
     // Se intancia el proxy
    VisaWSStub stub = new VisaWSStub();
     // Se le dice al proxy dónde puede encontrar el servicio web
    stub.setEndpoint("http://161.67.27.108:8988/soap/servlet/soaprouter");
     // Se llama al método ofertado por el proxy
    if (!stub.validar("", new Double(5.0)).booleanValue())
      throw new Exception("Operación no admitida");
}
```

Obsérvese que el tipo del segundo parámetro de la llamada a *validar* es de tipo *Double* (con mayúsculas) y no *double* (con minúsculas), a pesar de que el método toma un *double* en la clase en la que se encuentra implementado. Esto es así porque el generador del proxy ha optado por incluir aquel tipo de dato en lugar de éste.

Cuando el tipo de un parámetro o del resultado es un tipo complejo, es preciso realizar una descripción del tipo (o de la clase) en el documento WSDL. Supongamos que la clase *Visa* ofrece la siguiente operación:

public dominio.Tarjeta getTarjetaConMayorCredito()

La clase *dominio.Tarjeta* es un tipo complejo y debe especificarse en el documento que describe el servicio web. El entorno de desarrollo es capaz de añadir al WSDL la descripción estandarizada de esta clase:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--
Generated by the Oracle9i JDeveloper Web Services WSDL Generator
-->
- <!--
Date Created: Thu Apr 22 18:00:42 CEST 2004
-->
- <definitions name="VisaWS" targetNamespace="http://visa/dominio/Visa.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://visa/dominio/Visa.wsdl"
    xmlns:ns1="http://visa.dominio/IVisaWS.xsd">
  - <types>
    - <schema targetNamespace="http://visa.dominio/IVisaWS.xsd"
        xmlns="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
        ENC="http://schemas.xmlsoap.org/soap/encoding/">
      - <complexType name="dominio_Tarjeta" jdev:packageName="dominio"
          xmlns:jdev="http://xmlns.oracle.com/jdeveloper/webservices">
        - <all>
            <element name="numero" type="string" />
            <element name="titular" type="string" />
            <element name="validaDesde" type="string" />
            <element name="validaHasta" type="string" />
          </all>
        </complexType>
      </schema>
    </types>
    <message name="getTarjetaConMayorCredito0Request" />
  - <message name="getTarjetaConMayorCredito0Response">
      <part name="return" type="ns1:dominio_Tarjeta" />
    </message>
```

Figura 166. El WSDL incluye ahora la descripción del tipo complejo "Tarjeta"

3.03 Ejercicio práctico

Google ofrece un servicio web que permite ejecutar búsquedas en su buscador. Obviamente, el servicio está implementado en las máquinas de Google, pero puede construirse un cliente que utilice el servicio usando su documento WSDL.

Se pide que escriba un pequeño programa que invoque el servicio web de Google, de modo que posibilite la realización de búsquedas y que muestre los resultados en un frame o por la consola. También puede, aprovechando lo visto sobre aplicaciones web al principio de este capítulo, desarrollar una sencilla aplicación web que realice búsquedas en Google utilizando el servicio web en lugar de una llamada directa a la URL.

El documento WSDL está accesible en: http://api.google.com/GoogleSearch.wsdl. En http://api.google.com hay ejemplos de código y documentación adicional.

3.04 Ejercicio

Rediseñe el sistema mostrado en la Figura 103 utilizando servicios web.

3.05 Ejercicio

Rediseñe el sistema mostrado en la Figura 103 para que se permita el acceso tanto mediante RMI como mediante servicios web.

4. Introducción a las aplicaciones basadas en componentes distribuidos