## SISTEMA AVANZADO DE MENSAJERÍA USANDO WEB SERVICES

### Objetivo

El objetivo del presente trabajo es mostrar la misma aplicación implementada con dos APIs; Apahe SOAP y Apache Axis. De manera se espera que se logre visualizar las ventajas de trabajar con una y otra y las facilidades que prestan los editores que se encuentran en el mercado para el desarrollo de aplicaciones basadas en Web Services.

### Descripción de la aplicación a desarrollar

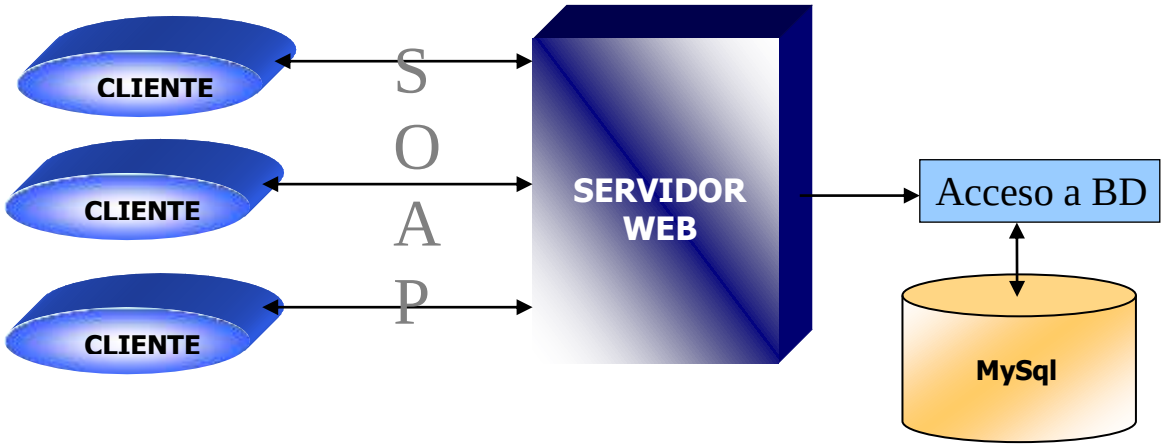
Se va a desarrollar un sistema de chat, el cual será controlado por un servidor que debe manejar una sesión por cada cliente que se conecte. Cada mensaje enviado por los clientes debe indicar la fuente del mismo. El cliente debe poder visualizar en su interfaz los logins de los usuarios que están activos en el chat, para que este pueda elegir a que usuario va a enviar un mensaje.

**Nota:**

- *Para poder ingresar al sistema un usuario debe poseer un login y password.*
- *El par de valores login , password debe almacenarse en una base de datos.*

### Diseño de la Arquitectura

La plataforma a utilizar en el lado del servidor es basada en JAVA. El hecho de utilizar SOAP trae como consecuencia que se pueda implementar un cliente en cualquier otro lenguaje.



## Implementación para el API Apache SOAP

### 1. Configuración de la plataforma

Se necesita un servidor web que soporte servlets. Para ello se va a usar Resin Web Server. Los archivos de instalación se pueden obtener de http://www.caucho.com/download/index.xtp . Básicamente, los pasos para la instalación son los siguientes:

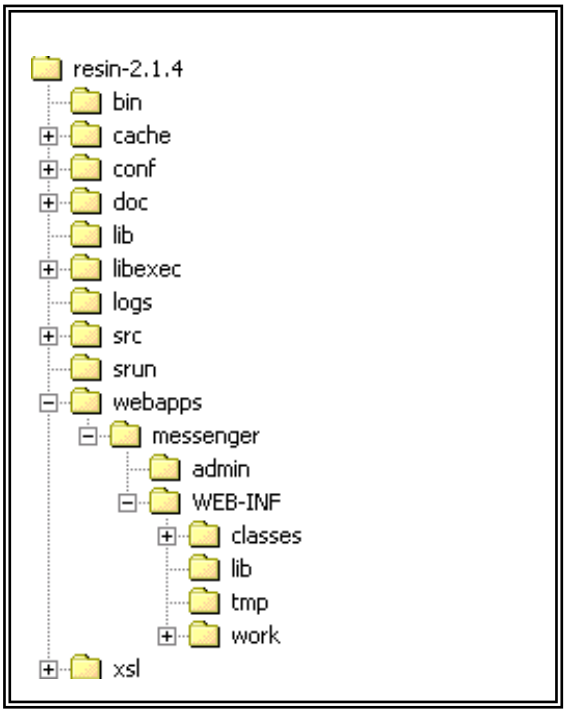
- Instalar el JDK 1.2 o superior.
- Descomprimir resin-2.1.4.zip
- Ejecutar resin-2.1.4/bin/httpd

Después confirmamos la correcta instalación abriendo el explorador con la dirección http://localhost:8080. Si queremos modificar la configuración del servidor, lo podemos hacer por medio del archivo resin.conf. En él podemos modificar aspectos como el puerto http.

Ahora, necesitamos contar con una implementación de la especificación SOAP para java. Vamos a usar la implementación del Apache SOAP. Apache SOAP puede ser usado como una librería cliente para invocar servicios SOAP disponibles, o como una herramienta del lado del servidor para implementar servicios accesibles por medio de SOAP. Para la instalación de esta implementación se deben seguir las instrucciones que se encuentran en http://xml.apache.org/soap/docs/index.html.

Nuestra aplicación, tanto en el lado del cliente como en el lado del servidor debe contar con todas las librerías indicadas en el proceso de instalación del Apache Soap, dentro de una carpeta llamada lib.

En el servidor web, la estructura de directorios es la siguiente:




La carpeta "messenger" es la carpeta de nuestra aplicación. En la carpeta messenger/WEB-INF/classes/ se encuentra el código compilado de la aplicación. En la carpeta messenger/WEB-INF/lib/ se encuentran todas las librerías necesarias para la aplicación.

La configuración de los servlets se realiza en el archivo web.xml que se encuentra en el directorio messenger/WEB-INF/. En este archivo se especifican los nombres, la clase, y la ruta de acceso al servlet. En el caso del propio servlet de la implementación SOAP, este se denomina rpcrouter en su valor por defecto. Este nombre, tanto como la ruta de acceso, puede ser modificado por los desarrolladores de la aplicación.

## 2. El Deployment Descriptor

Para proveer información sobre los servicios que están disponibles se utilizan documentos XML llamados "deployment descriptors". Ellos proveen una gran cantidad de información tal como la URN para el servicio (la cual es usada para

enrrutar la solicitud), detalles sobre los métodos y las clases si el servicio esta implementado en una clase java, o el nombre del script si el servicio esta implementado en un BSF que soporte lenguajes de scripting. El contenido del deployment descriptor depende del tipo de objeto que va a ser expuesto por medio de SOAP.

Para una clase java como la de la implementación realizada en este trabajo, el deployment es de la siguiente forma:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
      id="messenger">
    <isd:provider type="java" scope="Session" methods="iniciarSesion
         cerrarSesion sendReceive agregarUsuario">
       <isd:java class="messenger.server.ServidorSOAP"
        static="false" />
    </isd:provider>
    <isd:faultListener>org.apache.soap.server.DOMFaultListener
    </isd:faultListen>

 </isd:service>
```

En este caso **messenger** es la URN asignada al servicio**.** Los métodos expuestos son iniciarSesion, CerrarSesion, sendReceive y agregarUsuario. Messenger.server.ServidorSOAP es el nombre de la clase que implementa los métodos expuestos. En el elemento <java>, hay un atributo llamadlo "static" fijado a false indicando que los métodos expuestos no son estáticos. El elemento <provider> posee un atributo llamado "scope" que indica el "tiempo de vida" de la instanciación de la clase. En este caso se a fijado a "Session" que indica que el objeto durará lo que dure la sesión http.

3. **Implementacion de la interfaz SOAP**

Para llevar a cabo la implementación de la interfaz SOAP se desarrolló una clase llamada soapUtil.java. Esta es una clase utilitaria para manejar llamados SOAP. Básicamente, esta clase lleva a cabo las siguientes funciones:

- Crea el objeto Call, que es la interfaz principal del código SOAP RPC.
- Fija la URI objetivo en el objeto Call por medio del método setTargetURI(), usando la URN especificada por el deployment descriptor para identificarse a si mismo.
- Fija el nombre del método que se desea invocar en el objeto Call usando el método setMethodName(). Este método debe estar expuesto en el deployment descriptor del servicio que ha sido identificado por la URN.
- Crea los objetos "parámetro" necesarios para la llamada RPC y los fija en el objeto Call usando el método setParams(). El numero tipo de parámetros debe ser igual a los definidos por el servicio.
- Ejecuta el método invoke() del objeto Call y captura el objeto Response retornado por el método. El método invoke() tiene como parámetro la URL del endpoint donde se encuentra el servicio.
- Revisa el objeto Response para ver si se han generado errores usando el método generatedFault().
- Si se ha generado un error lo recupera usando el método getFault(), de lo contrario extrae el resultado o parámetros retornados por medio de los métodos getReturnValue() y getParams() respectivamente.

**4. Base de datos**

Se va a trabajar con una base de datos My SQL. EL driver usado es el mysql-connector-java-2.0.14-bin.jar. La base de datos almacena la información de los usuarios necesaria para realizar la validación. También almacena los mensajes enviados a los usuarios con la información necesaria para su direccionamiento (de quien, para quien, etc).

***Desarrollo***

La clase ServidorSOAP implementa los siguientes métodos:

- public String[] iniciarSesion(String usuario,String password)
  - retorna los usuarios conectados en el momento de la conexión.

- Private int validarUsuario(String nombre,String password)
  - Método que busca un usuario en la base de datos del sistema

- public String[] sendReceive(String who,String msg)
  - retorna los mensajes que el usuario a quien le escribe le mandó anteriormente.
  - En el caso de que msg sea el String vacío este método solo retorna los mensajes que le hayan sido enviados a "who".

- public void cerrarSesion()
  - Este método se debe invocar al final de la sesión para mantener coherente el estado de las sesiones. En caso de no ser invocado, la variable que contiene el numero de usuarios conectados almacena un valor erróneo.

En la siguiente tabla se encuentra el código fuente de la clase que implementa el Servicio Web.

```
package messenger.server;

import messenger.control.BeanConector;
import java.util.*;//import para trabajar con la clase Vector
import java.io.*;//import para la excepción IOException
import java.sql.*;//impot para el la utilizacion de sentencias SQL
import org.gjt.mm.mysql.Driver;

public class ServidorSOAP {
  /**
  * Constructor de la clase ServidorSOAP
```

```
  */
 public ServidorSOAP() {

   conectarBD();
 }
/**
*metodo que busca un usuario en la base de datos del sistema
*@param nombre, login del usuario
*@param password, password del usuario
*@return true si lo encuentra
*/

 private int validarUsuario(String nombre,String password){

    //se conecta al BD
    int x=con.conectar();
    if(x==-1)return -1;

    String consulta="select * from Usuarios where login= '"+nombre+"' and "+
                "password='"+password+"' ";

    con.consultar(consulta);//hace la consulta

    if(con.siguiente())//si lo encontró
    return 1;
    return -1;

  }


  /*Funcion que establece la conexión con la base de datos*/
 private int conectarBD(){
     con.setDriver("org.gjt.mm.mysql.Driver");
     con.setUrl("jdbc:mysql://localhost/messenger");
```

```
        con.setLogin("root");
        con.setPassword("");
        int x=con.conectar();

        return x;
    }

    private String password
    private BeanConector con= new BeanConector();
    private int estado;

    public String[] iniciarSesion(String nombre,String password) throws Exception{

      this.password=password;
      int b=validarUsuario(nombre,password);

      if(b==-1){
        String error="0000";
        ArrayList users = new ArrayList();
        users.add(error);
        return getStringArray(users);


      }//si el usuario hace parte de los usuarios del sistema, lo conecta
      else if(b==1){


//*****************************************************************

       String consulta="select estado from Usuarios where login= '"+nombre+"' and "+
                  "password='"+password+"' ";

      con.consultar(consulta);//hace la consulta
      while(con.siguiente()){//si lo encontró
```

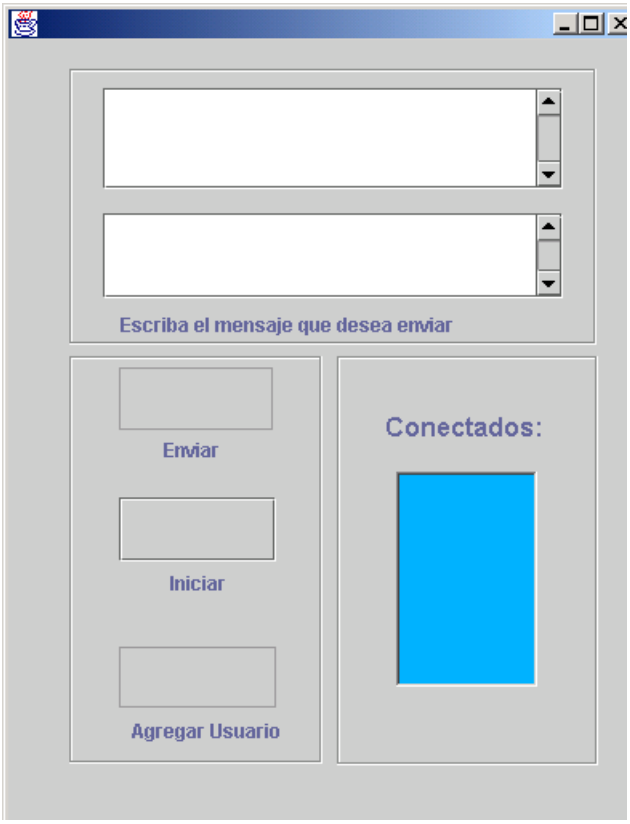
```
    estado = con.getEntero("estado");
    }
    if(estado==0)
    {
     String actualizacion="UPDATE usuarios SET estado=1 WHERE login='"+ nombre
+"'";
     con.actualizar(actualizacion);

    }
    else if(estado==1)
    {
     String error="1111";
     ArrayList users = new ArrayList();
     users.add(error);
     return getStringArray(users);

    }
   }

     /*Si no encuentra el login con su password asociado pide al cliente que
     rectifique estos datos*/
     else{
       String error="1010";
       ArrayList users = new ArrayList();
       users.add(error);
       return getStringArray(users);

     }
     String[] respuesta=retornarConectados(nombre);
     return respuesta;

  }

  public String[] retornarConectados(String n)
```

```
  {
      String cta="select login from usuarios where estado='1'";
      ArrayList conect= new ArrayList();
      con.consultar(cta);
      int i=0;
      while(con.siguiente()){
        i++;
        String conectado = con.getCadena("login");
        if (!conectado.equalsIgnoreCase(n))
            conect.add(conectado);
      }
     return getStringArray(conect);


  }
 /**
  * Este metodo se debe invocar al final de la sesion del usuario para
  * mantener coherente el estado de las sesiones
  * @return retorna los ids de las personas que me están intentando contactar al
momento
  * de mi retiro
  * @throws Exception
  */
 public boolean cerrarSesion(String nombre) throws Exception{


    String actualizacion="UPDATE usuarios SET estado=0 WHERE login='"+ nombre +"'";
    con.actualizar(actualizacion);
    con.cerrarCx();
    nombre = "";
    password="";
    return true;
 }
   public String[] sendReceive(String whom,String msg, String nombre) throws
Exception{
     /*
```

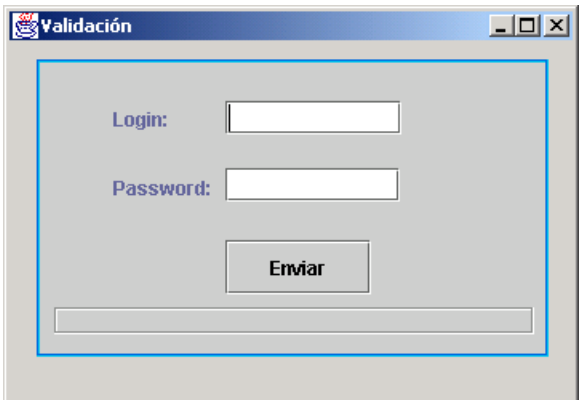
```
    Esta operacion debería ser implementada con una transaccion.
    Select+ Delete
    */
    con.conectar();
    String cta="select * from mensajes where whom='"+ nombre +"'";
    ArrayList res= new ArrayList();
    con.consultar(cta);
    while(con.siguiente()){
      String msgtome = con.getCadena("who")+"  dice: " + con.getCadena("msg");
      res.add(msgtome);
    }
    cta="delete from mensajes where whom='"+ nombre +"'";
    con.actualizar(cta);
    //Si es el mensaje vacío, solo quiere recibir
    if(msg!=null && !msg.equals("")){
      cta="insert into mensajes(who,whom,msg,time_stamp)" +
         " values('"+ nombre + "','"+ whom + "','"+ msg + "','"+ new
Timestamp(System.currentTimeMillis()).toString()+ "')";
      con.actualizar(cta);
    }
    return getStringArray(res);
  }
  private String[] getStringArray(ArrayList al){
    String[] res = new String[al.size()];
    for (int i = 0; i < al.size(); i++) {
      res[i] = (String)al.get(i);
    }
    return res;
  }
}
```

En el lado del cliente encontramos las clases Cliente, Validación , NuevoUsuario y Temporizador. Las tres primeras clases son interfaces graficas que me dan acceso a los diferentes servicios del chat como lo son la validación de usuarios, la creación de nuevos usuarios para el chat y el envío de mensajes. La clase Temporizador es un hilo

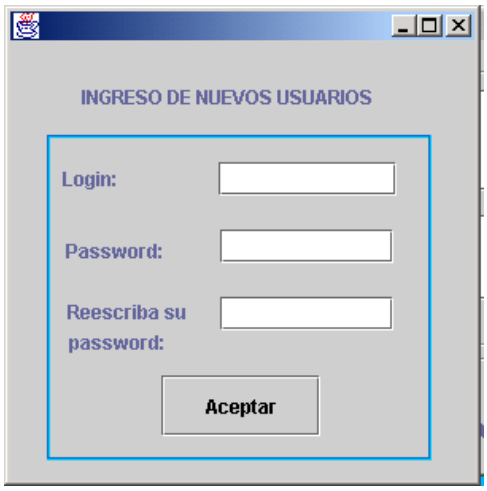
que se encarga de llamar cada 5 segundos al método SendReceive del ServidorSOAP para recibir los mensajes que le hayan enviado a ese usuario. Cuando el cliente corre la aplicación se despliega la siguiente interfaz:




Lo primero que el cliente debe hacer es iniciar sesión. Para hacerlo debe presionar el botón "Iniciar", en ese momento se le despliega la interfaz de validación:



Al introducir los datos y presionar enviar es llamado el método **iniciarSesion()** que se encarga de establecer la sesión con el servidor y validar los datos del usuario, retornando los usuarios que en ese momento estén conectados. Al mismo tiempo se crea el hilo temporizador que hace un llamado al método **sendReceive()** cada 5 segundos para verificar si han llegado mensajes para el usuario. Cuando un usuario desea enviar un mensaje a otro, escribe el mensaje en la ventana de texto y presiona "Enviar" en la interfaz principal. En este momento es llamado el método **sendReceive()** que se encarga de almacenar el mensaje enviado en la base de datos y verificar si hay nuevos mensajes para el usuario que envió el mensaje. El usuario puede agregar nuevos usuarios a la aplicación al hacer click en "Agregar Usuario".

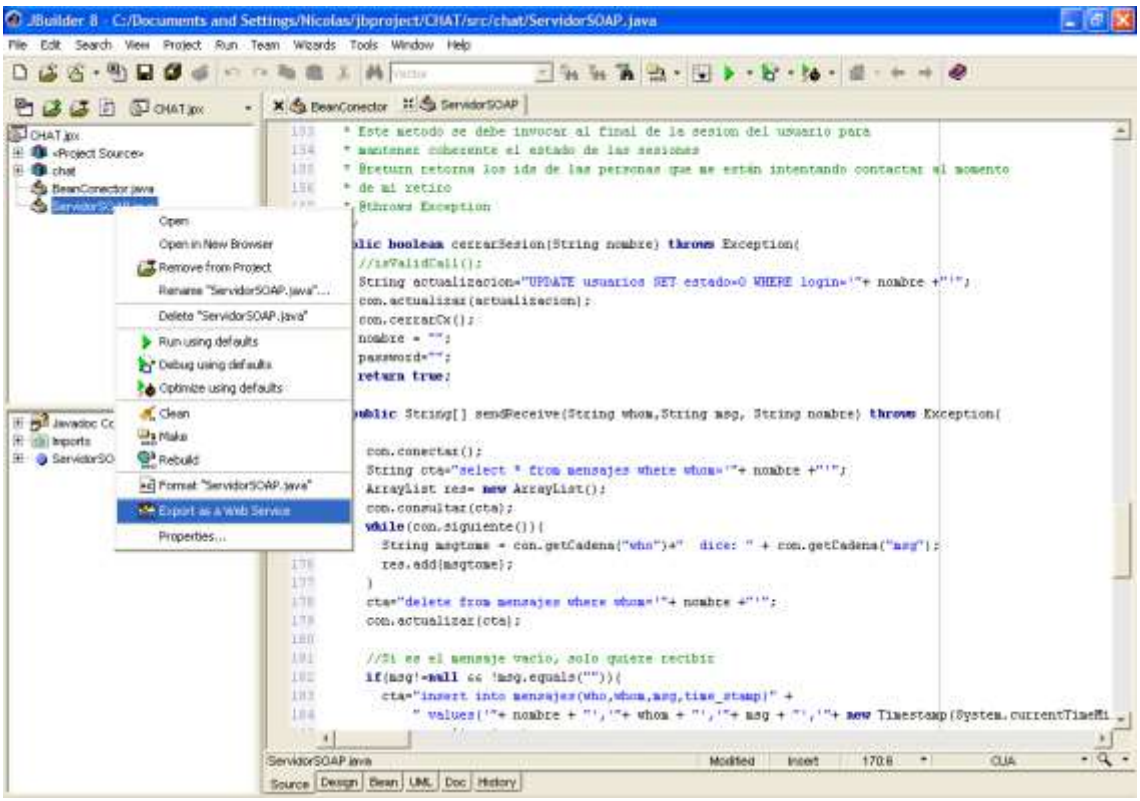



Después de llenar los datos y hacer clic en "Aceptar", es llamado el método **agregarUsuario()**, que se encarga de almacenar los datos del nuevo usuario en la base de datos.

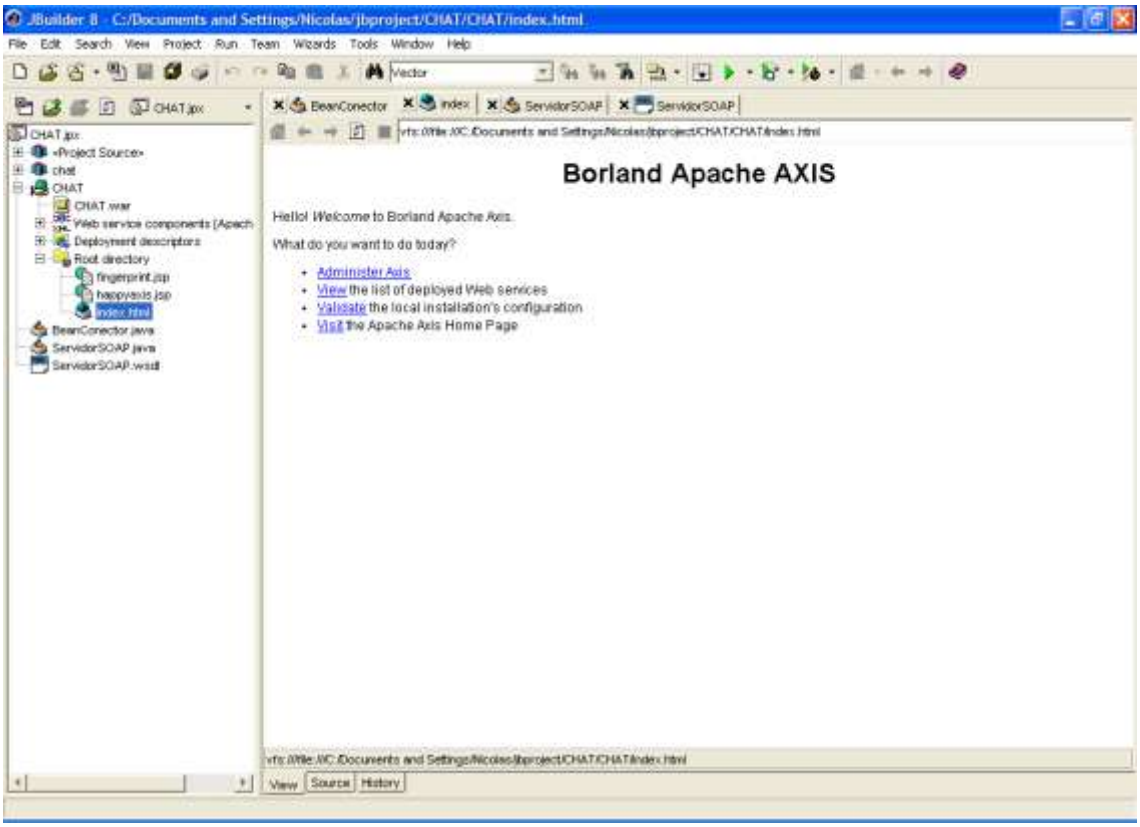
Finalmente, para cerrar la aplicación se debe llamar el método **finalizarSesion()** para terminar la sesión con el servidor Web. Hay que tener presente que todos los métodos en negrita se encuentran en el servidor y son llamados remotamente a través de SOAP por medio de la clase soapUtil.

**Implementación para el API Apache Axis**

El trabajo con esta API va a ser realizado a través del IDE Jbuilder 8 que incluye un Kit para el desarrollo de aplicaciones basadas en Web Services. Una ves que se tiene la clase que implementa la lógica del servidor (Se va a usar la clase desarrollada en la implementación para el Apache Soap), se hace uso del Wizard "Export as a Web Service". Esto se vería de la siguiente manera:




En ese momento , el IDE muestra una serie de interfaces en las que nos permite configurar detalles como el tipo de sesión que deseamos manejar (ya sea request, session o application), y otra serie de elementos. Una ves que le damos "finish", el IDE genera el documento WSDL que describe el servicio, los descriptores necesarios para desplegar el Servicio en el servidor Tomcat y el archivo de extensión war para montar al servidor. Además genera una serie de interfaces web que permiten verificar la correcta configuración del API apache Axis en el servidor y visualizar los Servicios Web desplegados en el mismo.




A continuación se muestra el documento WSDL generado por la herramienta.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://server.messenger"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://server.messenger-impl" xmlns:intf="http://server.messenger"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
  <schema targetNamespace="http://server.messenger"
xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
```

```
   <complexType name="ArrayOf_xsd_string">
    <complexContent>
     <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]"/>
     </restriction>
    </complexContent>
   </complexType>
   <element name="ArrayOf_xsd_string" nillable="true"
type="intf:ArrayOf_xsd_string"/>
  </schema>
 </wsdl:types>

   <wsdl:message name="retornarConectadosRequest">

      <wsdl:part name="n" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="iniciarSesionRequest">

      <wsdl:part name="nombre" type="xsd:string"/>

      <wsdl:part name="password" type="xsd:string"/>

   </wsdl:message>

   <wsdl:message name="sendReceiveResponse">

      <wsdl:part name="sendReceiveReturn" type="intf:ArrayOf_xsd_string"/>

   </wsdl:message>

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

```
    <wsdl:part name="retornarConectadosReturn"
type="intf:ArrayOf_xsd_string"/>

  </wsdl:message>

  <wsdl:message name="cerrarSesionRequest">

    <wsdl:part name="nombre" type="xsd:string"/>

  </wsdl:message>

  <wsdl:message name="iniciarSesionResponse">

    <wsdl:part name="iniciarSesionReturn" type="intf:ArrayOf_xsd_string"/>

  </wsdl:message>

  <wsdl:message name="cerrarSesionResponse">

    <wsdl:part name="cerrarSesionReturn" type="xsd:boolean"/>

  </wsdl:message>

  <wsdl:message name="sendReceiveRequest">

    <wsdl:part name="whom" type="xsd:string"/>

    <wsdl:part name="msg" type="xsd:string"/>

    <wsdl:part name="nombre" type="xsd:string"/>

  </wsdl:message>

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

```
    <wsdl:operation name="iniciarSesion" parameterOrder="nombre password">

      <wsdl:input message="intf:iniciarSesionRequest"
name="iniciarSesionRequest"/>

      <wsdl:output message="intf:iniciarSesionResponse"
name="iniciarSesionResponse"/>

    </wsdl:operation>

    <wsdl:operation name="retornarConectados" parameterOrder="n">

      <wsdl:input message="intf:retornarConectadosRequest"
name="retornarConectadosRequest"/>

      <wsdl:output message="intf:retornarConectadosResponse"
name="retornarConectadosResponse"/>

    </wsdl:operation>

    <wsdl:operation name="cerrarSesion" parameterOrder="nombre">

      <wsdl:input message="intf:cerrarSesionRequest"
name="cerrarSesionRequest"/>

      <wsdl:output message="intf:cerrarSesionResponse"
name="cerrarSesionResponse"/>

    </wsdl:operation>

    <wsdl:operation name="sendReceive" parameterOrder="whom msg nombre">

      <wsdl:input message="intf:sendReceiveRequest"
```

```
name="sendReceiveRequest"/>

        <wsdl:output message="intf:sendReceiveResponse"
name="sendReceiveResponse"/>

      </wsdl:operation>

   </wsdl:portType>

   <wsdl:binding name="ServidorSOAPSoapBinding" type="intf:ServidorSOAP">

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

      <wsdl:operation name="iniciarSesion">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="iniciarSesionRequest">

            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://server.messenger" use="encoded"/>

         </wsdl:input>

         <wsdl:output name="iniciarSesionResponse">

            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://server.messenger" use="encoded"/>

         </wsdl:output>
```

```
    </wsdl:operation>

    <wsdl:operation name="retornarConectados">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="retornarConectadosRequest">

        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://server.messenger" use="encoded"/>

      </wsdl:input>

      <wsdl:output name="retornarConectadosResponse">

        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://server.messenger" use="encoded"/>

      </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="cerrarSesion">

      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="cerrarSesionRequest">

        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://server.messenger" use="encoded"/>
```

```
        </wsdl:input>

        <wsdl:output name="cerrarSesionResponse">

          <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://server.messenger" use="encoded"/>

        </wsdl:output>

      </wsdl:operation>

      <wsdl:operation name="sendReceive">

        <wsdlsoap:operation soapAction=""/>

        <wsdl:input name="sendReceiveRequest">

          <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://server.messenger" use="encoded"/>

        </wsdl:input>

        <wsdl:output name="sendReceiveResponse">

          <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://server.messenger" use="encoded"/>

        </wsdl:output>

      </wsdl:operation>
```

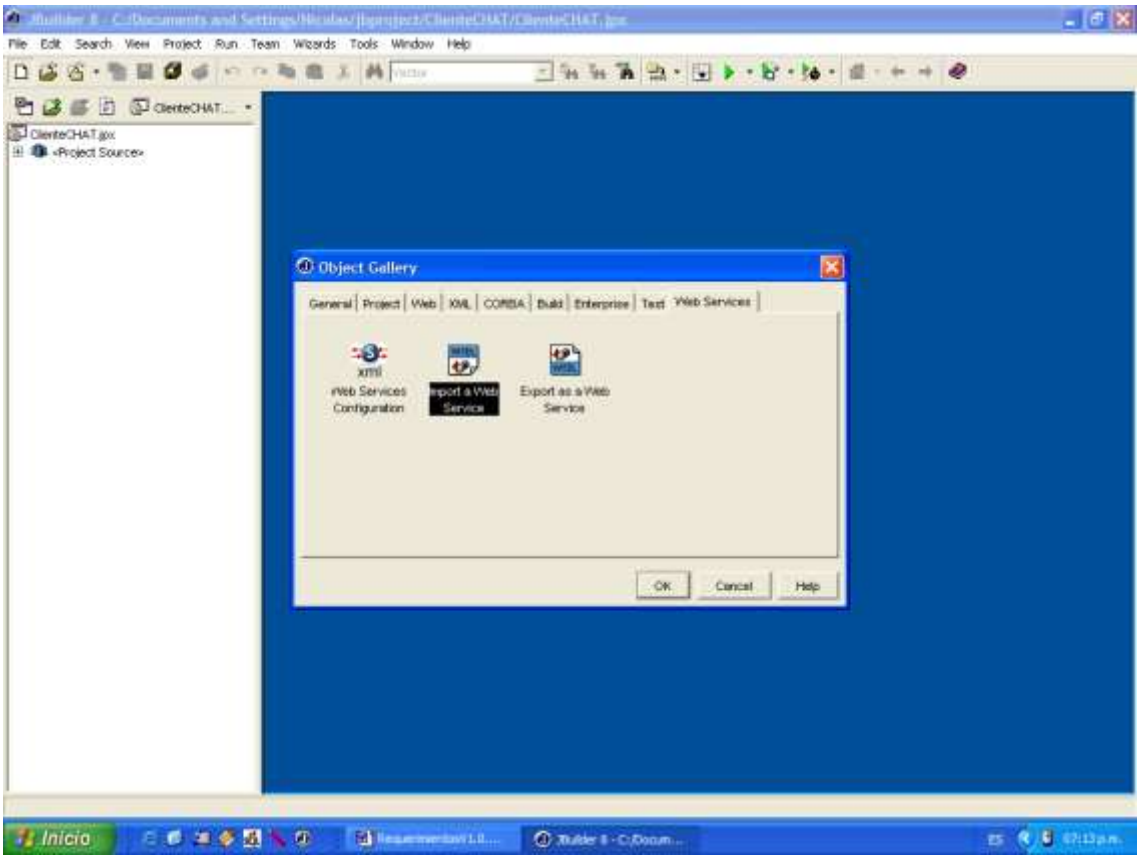
```
  </wsdl:binding>

  <wsdl:service name="ServidorSOAPService">

    <wsdl:port binding="intf:ServidorSOAPSoapBinding" name="ServidorSOAP">

      <wsdlsoap:address
location="http://localhost:8080/MensajeriaWS/services/ServidorSOAP"/>

    </wsdl:port>

  </wsdl:service>

</wsdl:definitions>
```

Una ves que el servidor Tomcat es inicializado, el Servicio CHAT, esta listo para recibir peticiones de los clientes.

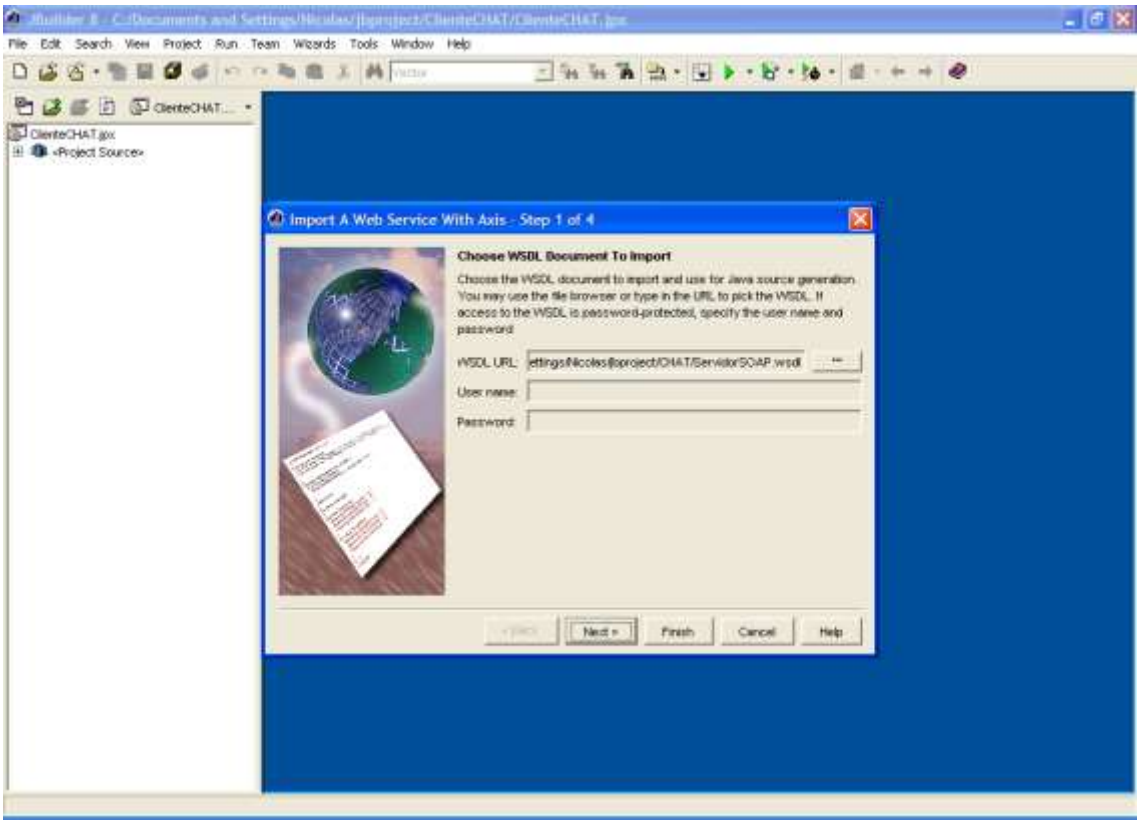
**Construcción del Cliente**

Para la construcción del cliente, el IDE también trae un wizard que genera automáticamente a partir del documento WSDL , las clases necesarias para la invocación del servicio Web. Para hacer esto, procedemos de la siguiente manera:

- Vamos a new , a la pestaña Web Services y seleccionamos Import a WSDL



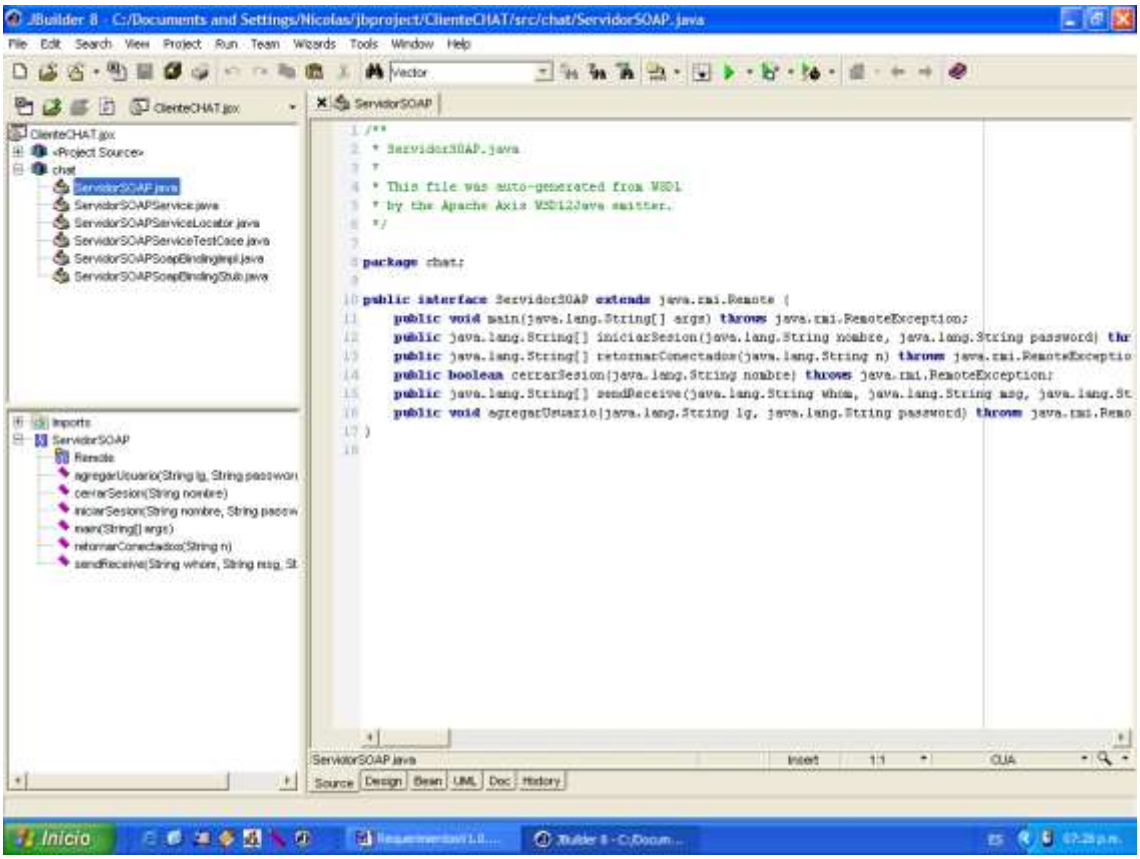

- En ese momento nos aparece una interfaz que nos permite seleccionar el documento WSDL.




En ese momento se generan las clases cliente:

- ServidorSOAP.java
- ServidorSOAPService.java
- ServidorSOAPServiceLocator.java
- ServidorSOAPSoapBindingImpl.java
- ServidorSOAPSoapBindingStub.java

Estas clases me permiten de una manera muy fácil realizar las invocaciones al Servicio Web CHAT. Para ayudar en la implementación del cliente, el IDE genera una clase de prueba llamada ServidorSOAPSoapTestCase.java que implementa la invocación de cada uno de los métodos del servicio web. Tomando esta clase como ejemplo podemos desarrollar los clientes a la medida de nuestras necesidades, a través de las clases auxiliares generadas por el mismo IDE.




Como podemos observar el IDE facilita muchísimo el desarrollo de aplicaciones basadas en Servicios Web, encargándose de toda la parte de los meta datos, a nuestro juicio la mas complicada en el proceso de desarrollo. También es importante notar que el API Apache Axis soporta la especificación de WSDL, no soportada por el API Apache SOAP.

De esta manera esperamos haya quedado claro, el proceso de creación de un servicio Web con las dos APIs.