Spago
Web Service Guide

Authors Angelo Bernabei
Gianfranco Boccalon
Andrea Zoppello

INDEX

Document Goals

This document contains some guidelines relative to the configuration of the projects built with Spago, using Eclipse, JBoss and Lomboz. You'll find detailed information about the publishing of services accessible by the SOAP protocol.

Version History

Version/Release n° :	1.1	**Date Version/Release** :	29/10/2004
Description:	Syntax correction		
Version/Release n° :	1.2	**Date Version/Release** :	10/01/2006
Description:	Added the Axis Implementation		

1 Available Implementations

1.1 Apache SOAP

The Apache SOAP implementation is the first integrated in Spago. This implementation is simple to use but it's quite old: it's not able to support some latest SOAP specifications, for example the attachments.

1.1.1 Server Configuration

In this section with the term "server" we mean a (Spago) application that exposes services on the SOAP channel. In this section we only refers to webservices exposed on the http endpoint, so the "server" is a web application properly configured to handle SOAP request. The term <CONTEXT-ROOT> identify the context of the application that is the SOAP server.

The sample application SpagoStartup is already configured to make its services accessible as Web Services, so you can refer to this application regarding the libraries and the configuration files.

To configure properly the server is necessary:

1. Ensure that the following libraries are present on the folder <CONTEXT-ROOT>/WEB-INF/lib, and if they're not there take it from Spago distributions.
    - soap-2.3.jar
    - activation.jar
    - mail.jar
    - spago-core-X.X.jar
    - spago-web-X.X.jar
2. Add the configurations of the servlets and of servlet-mappings of Apache SOAP adding the following line to the file <CONTEXT-ROOT>/WEB-INF/web.xml.

```
<context-param>
  <param-name>ConfigFile</param-name>
  <param-value>/WEB-INF/conf/soap/soap.xml</param-value>
</context-param>
<servlet>
  <servlet-name>rpcrouter</servlet-name>
  <display-name>Apache-SOAP RPC Router</display-name>
  <description>no description</description>
  <servlet-class>org.apache.soap.server.http.RPCRouterServlet</servlet-class>
  <init-param>
       <param-name>faultListener</param-name>
```

```
            <param-value>org.apache.soap.server.DOMFaultListener</param-value>
        </init-param>
    </servlet>
    <servlet>
        <servlet-name>messagerouter</servlet-name>
        <display-name>Apache-SOAP Message Router</display-name>
        <description>no description</description>
        <servlet-class>org.apache.soap.server.http.MessageRouterServlet</servlet-
class>
        <init-param>
            <param-name>faultListener</param-name>
            <param-value>org.apache.soap.server.DOMFaultListener</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>rpcrouter</servlet-name>
        <url-pattern>/servlet/rpcrouter</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>messagerouter</servlet-name>
        <url-pattern>/servlet/messagerouter</url-pattern>
    </servlet-mapping>
```

3. Copy the folder /WEB-INF/conf/soap supplied by Spago, in the folder <CONTEXT-ROOT>/WEB-INF/conf (in the folder "conf" should be present the folders "spago" and "soap"). This folder contains the files soap.xml and soap.ds which is a binary file containing the description of the SOAP services ("ds" means "deployed services"). This file is editable through the SOAP administration page, accessible at the URL <CONTEXT-ROOT>/admin.

4. Copy the folder WEB-INF/wsdl, supplied by Spago, in the folder <CONTEXT-ROOT>/WEB-INF/wsdl. Also if this folder is missing the Adapter works properly.

1.1.2 Client configuration

To use the SOAP services you can use a Spago class named AdapterSOAPProxy.

Please read the paragraph [3.Service Publications as Webservice] for detailed information about SOAP clients.

1.2 Apache Axis

As we saw in the previous section with Spago it is possible to access business services (ACTION or PAGE) using a webservice on the SOAP channel.

The problem of the SOAP specification is that there are some problems that are not handled automatically by the protocol. The most common problem with soap webservices is the handling of applicative sessions within webservice calls.

For this reason in Spago the web service AdapterAxis has been introduced. This adapter exposes Spago services on the SOAP channel using the features of the Apache Axis SOAP stack (http://ws.apache.org/axis/), that provides some mechanism to implement the functionality of the Spago SessionContainer.

So when this webservice is used we can write SessionContainer aware Spago services, without problems.

1.2.1 Server Configuration

In this section with the term "server" we mean a (Spago) application that exposes services on the SOAP channel. In this section we only refers to webservices exposed on the http endpoint, so the "server" is a web application properly configured to handle soap request. The term <CONTEXT-ROOT> identify the context of the application that is the soap server.

To configure properly the server is necessary:

1. Ensure that the following libraries are present on the folder <CONTEXT-ROOT>/WEB-INF/lib, and if they're not there take it from Axis 1.2.1 and Spago distributions.
    - axis.jar
    - axis-ant.jar
    - commons-discovery-0.2.jar
    - commons-logging-1.0.4.jar
    - jaxrpc.jar
    - log4j-1.2.8.jar
    - saaj.jar
    - wsdl4j-1.5.1.jar
    - spago-axis-X.X.jar
    - spago-core-X.X.jar
    - spago-web-X.X.jar
2. Add the configurations of the servlets and of servlet-mappings of axis adding the following line to the file <CONTEXT-ROOT>/WEB-INF/web.xml.

```
<servlet>
        <display-name>
```

```
        Apache-Axis Servlet</display-name>
        <servlet-name>AxisServlet</servlet-name>
        <servlet-class>
        org.apache.axis.transport.http.AxisServlet</servlet-class>
    </servlet>
    <servlet>
        <display-name>
        Axis Admin Servlet</display-name>
        <servlet-name>AdminServlet</servlet-name>
        <servlet-class>
        org.apache.axis.transport.http.AdminServlet</servlet-class>
        <load-on-startup>100</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/servlet/AxisServlet</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>*.jws</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>AdminServlet</servlet-name>
        <url-pattern>/servlet/AdminServlet</url-pattern>

    </servlet-mapping>
```

3. Copy the file server-config.wsdd provided with spago-axis in the folder <CONTEXT-ROOT>/WEB-INF/ (at the same level of web.xml).

4. Copy the file AdapterAxis.wsdl in the folder <CONTEXT-ROOT>/WEB-INF/wsdl.

1.2.2 Client Configuration

A soap client can be a standalone application or a web application. In both cases in this and in the next section we refers to clients written using Java Tecnology.

In both case the configuration is very simple:

1. Ensure that the following libraries are present on application classpath (<CONTEXT-ROOT>/WEB-INF/lib for web application), and if they're not there take it from Axis 1.2.1 and Spago distributions.

    - axis.jar
    - axis-ant.jar
    - commons-discovery-0.2.jar
    - commons-logging-1.0.4.jar

- jaxrpc.jar
- log4j-1.2.8.jar
- saaj.jar
- wsdl4j-1.5.1.jar
- spago-axis-X.X.jar
- spago-core-X.X.jar
- spago-web-X.X.jar

2. Copy the file client-config.wsdd provide with spago-axis on the application class path.

1.2.3 AdapterAxisProxy usage and session handling

With Spago it is possible to invoke directly the AdapterAxis webservice without know SOAP details. This is done with the utility class AdapterAxisProxy.

This class provide the method *public String service(String request)* that takes the XML of the service request as input and the method *public Long getSessionId()* to obtain the session identifier generated by the request.

The following code shows the use case when only one proxy instance is used to make multiple calls to the webservice. In that case the session handling is automatically done by Axis.

```
public static void main(String[] args) {


        //
        // Example1 : Use the same proxy instance for mutiple calls.
        //                 In the case you'use the same proxy you doesnt't
        //                 need to handle manually the session ID
        AdapterAxisProxy proxy1 = null;


        StringBuffer aStringBuffer = new StringBuffer();
        aStringBuffer.append("<SERVICE_REQUEST ACTION_NAME=\"INCREMENTA_NUMERO\" />");
        try{
              proxy1 = new AdapterAxisProxy();
              proxy1.setEndpoint("http://localhost:8080/ValidationTest/services/AdapterAxis");
```

```
            String response = proxy1.service(aStringBuffer.toString());
            System.out.println(" Response = ["+response+"]");
            Thread.sleep(1000);

            response = proxy1.service(aStringBuffer.toString());
            System.out.println(" Response = ["+response+"]");
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
```

In the second example we show the use case where two instances of the proxy are used to call the webservices and we want to use the same session. In that case the session identifier is retrieved after the first request and passed to the second proxy.

```
//
// Example2 : Using two instance of proxy you need to pass the session id,
//                    returned from proxy1 when constructing proxy2 instance
public static void main(String[] args) {


        AdapterAxisProxy proxy1 = null;
        AdapterAxisProxy proxy2 = null;
        StringBuffer aStringBuffer = new StringBuffer();
        aStringBuffer.append("<SERVICE_REQUEST ACTION_NAME=\"INCREMENTA_NUMERO\" />");
        try{
              proxy1 = new AdapterAxisProxy();
              proxy1.setEndpoint("http://localhost:8080/ValidationTest/services/AdapterAxis");
              String response = proxy1.service(aStringBuffer.toString());
              System.out.println(" Response = ["+ response +"]");
              Long idSession = proxy1.getSessionId();
              Thread.sleep(1000);
              proxy2 = new AdapterAxisProxy(idSession);
              proxy2.setEndpoint("http://localhost:8080/ValidationTest/services/AdapterAxis");
              response = proxy2.service(aStringBuffer.toString());
              System.out.println(" Response = ["+response+"]");

        }catch (Exception e) {
              e.printStackTrace();
        }
   }
```

2 An example

The first step is the creation of a JSP page and an action that implement a multiplier by 2 of a number.

Following the steps to create the sample:

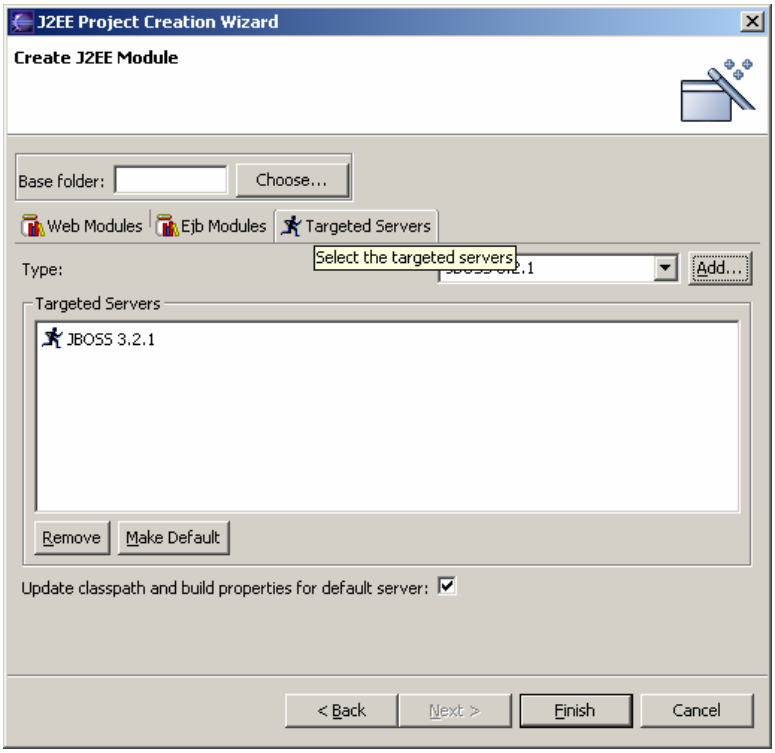
Step 1: create a Lomboz project. Select the item *File.new* on the Eclipse menu and choose the item voce *Project*. In the window that will appear you can choose the project type.

Step 2: Choose the type *Lomboz J2EE Wizards* and then select *Lomboz J2EE Project*, then choose the "Next" button.

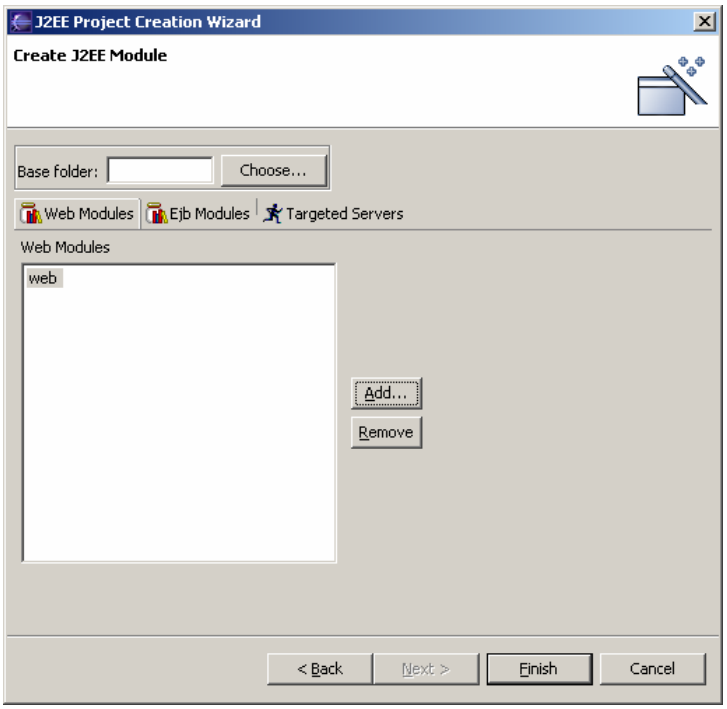
Step 3: Write the project name and select "Next".

Step 4: Keep the default java settings and choose "Next".

Step 5: In the Lomboz settings, choose the server (previously configured in the Lomboz installation),

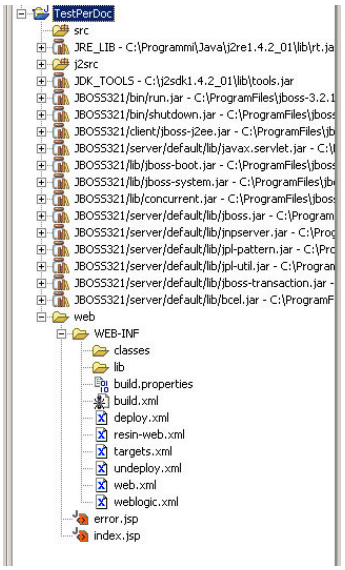


And add a Web module that represents the web context of the application.

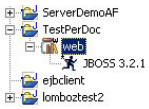


Select the "Finish" button to create the project.

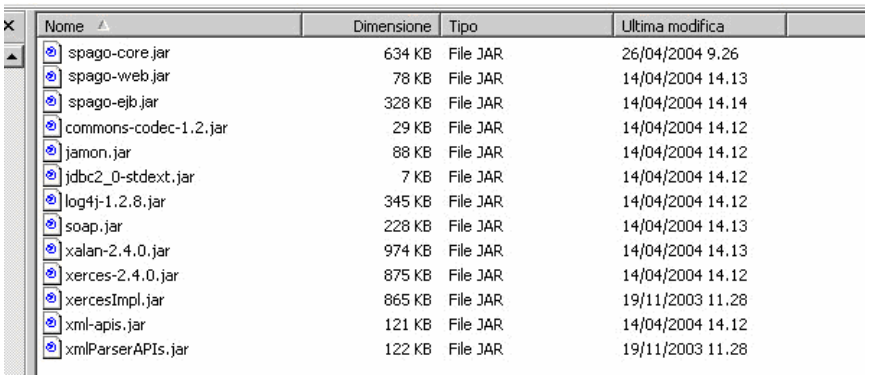
You'll find a project with the following structure:



As you can see Lomboz uses Ant for the deploy operations on JBoss. Once created the project, in the Lomboz J2EE View you can invoke the deploy/undeploy operations and start and stop JBoss.



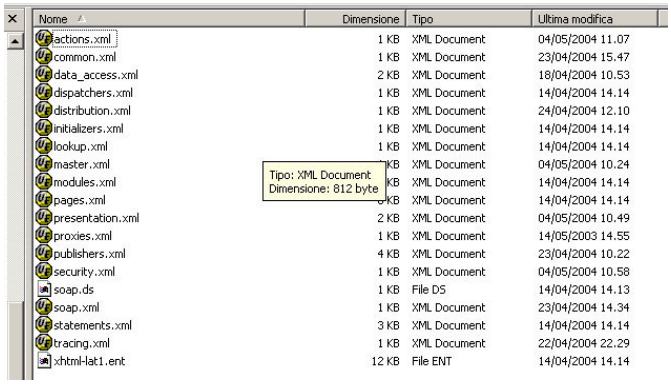
You can now add the Spago libraries (in the folder /web/WEB-INF/lib) and all the configuration files:



Nome	Dimensione	Tipo	Ultima modifica
spago-core.jar	634 KB	File JAR	26/04/2004 9.26
spago-web.jar	78 KB	File JAR	14/04/2004 14.13
spago-ejb.jar	328 KB	File JAR	14/04/2004 14.14
commons-codec-1.2.jar	29 KB	File JAR	14/04/2004 14.12
jamon.jar	88 KB	File JAR	14/04/2004 14.12
jdbc2_0-stdext.jar	7 KB	File JAR	14/04/2004 14.12
log4j-1.2.8.jar	345 KB	File JAR	14/04/2004 14.12
soap.jar	228 KB	File JAR	14/04/2004 14.13
xalan-2.4.0.jar	974 KB	File JAR	14/04/2004 14.13
xerces-2.4.0.jar	875 KB	File JAR	14/04/2004 14.12
xercesImpl.jar	865 KB	File JAR	19/11/2003 11.28
xml-apis.jar	121 KB	File JAR	14/04/2004 14.12
xmlParserAPIs.jar	122 KB	File JAR	19/11/2003 11.28

Remember also to add the libraries to the project classpath.

The folder /web/WEB-INF/conf/spago should contain at least the following items.



Nome	Dimensione	Tipo	Ultima modifica
actions.xml	1 KB	XML Document	04/05/2004 11.07
common.xml	1 KB	XML Document	23/04/2004 15.47
data_access.xml	2 KB	XML Document	18/04/2004 10.53
dispatchers.xml	1 KB	XML Document	14/04/2004 14.14
distribution.xml	1 KB	XML Document	24/04/2004 12.10
initializers.xml	1 KB	XML Document	14/04/2004 14.14
lookup.xml	1 KB	XML Document	14/04/2004 14.14
master.xml	KB	XML Document	04/05/2004 10.24
modules.xml	KB	XML Document	14/04/2004 14.14
pages.xml	KB	XML Document	14/04/2004 14.14
presentation.xml	2 KB	XML Document	04/05/2004 10.49
proxies.xml	1 KB	XML Document	14/05/2003 14.55
publishers.xml	4 KB	XML Document	23/04/2004 10.22
security.xml	1 KB	XML Document	04/05/2004 10.58
soap.ds	1 KB	File DS	14/04/2004 14.13
soap.xml	1 KB	XML Document	23/04/2004 14.34
statements.xml	3 KB	XML Document	14/04/2004 14.14
tracing.xml	1 KB	XML Document	22/04/2004 22.29
xhtml-lat1.ent	12 KB	File ENT	14/04/2004 14.14

Tipo: XML Document
Dimensione: 812 byte

Substitute the file web.xml with the file provided by Spago.

Change the directory where Ecplise produce the .class files, to the folder */web/WEB-INF/classes*. You can do it from the project properties windows.

Please read the Spago documentation for detailed information of configuration files.

Define the test service changing the following files:

action.xml

```
<ACTION class="it.eng.spago.test.Sommatore" distributed="FALSE" name="Sommatore" scope="REQUEST">
    <CONFIG />
</ACTION>
```

presentation.xml

```
<MAPPING business_name="Sommatore" business_type="ACTION" publisher_name="SommatoreP" />
```

publisher.xml

```
<PUBLISHER name="SommatoreP">
    <RENDERING channel="HTTP" mode="FORWARD" type="JSP">
        <RESOURCES>
            <ITEM prog="0" resource="/index.jsp" />
        </RESOURCES>
    </RENDERING>
```

```
</PUBLISHER>
```

This is the class that implements the service:

```
package it.eng.spago.test;

import it.eng.spago.base.SourceBean;
import it.eng.spago.dispatching.action.AbstractAction;

/**
 * @author bernabei
 */
public class Sommatore extends AbstractAction{


	public void service(SourceBean request, SourceBean response) throws Exception {
		String primo=(String)request.getAttribute("primo");
		String secondo=(String)request.getAttribute("secondo");
		int risultato=Integer.parseInt(primo)+ Integer.parseInt(secondo);
		ValueObjectTest vo=new ValueObjectTest();
		vo.setMsg(Integer.toString(risultato));
		response.setAttribute("risultato",vo);
	}
}
```

Following a utility class:

```
package it.eng.spago.test;

/**
 * @author bernabei
 */
public class ValueObjectTest {
private String msg="";
/**
 * @return
 */
public String getMsg() {
	return msg;
}
```

```
/**
 * @param string
 */
public void setMsg(String string) {
        msg = string;
}
public String toString(){
        return msg;
}
}
```

The result will be rendered by this JSP page:

```
<%@ page
                        contentType="text/html; charset=ISO-8859-1"
                        pageEncoding="ISO-8859-1"
                        session="true"
                        import="it.eng.spago.base.*,it.eng.spago.test.ValueObjectTest"
%>

<%
ValueObjectTest risultato=null;
ResponseContainer responseContainer=ResponseContainerAccess.getResponseContainer(request);
if (responseContainer!=null){
        SourceBean sb =         responseContainer.getServiceResponse();
        if (sb!=null) risultato = (ValueObjectTest)sb.getAttribute("risultato");
}
if (risultato==null) risultato=new ValueObjectTest();
%>
<html>
        <head>
                <title>Sommatore...</title>
        </head>
        <body>
        <form name="forem" method="post" action="./servlet/AdapterHTTP?ACTION_NAME=Sommatore">
                <center>SOMMATORE...</center>
                Primo Numero : <input type="text" name="primo">
                Secondo Numero : <input type="text" name="secondo">
                <input type="submit" class="submit" value="Somma">
                </form>
                Risultato= <input type="text"  name=”risultato” value='<%=risultato.getMsg()%>'>
        </body>
</html>
```

Now you can test the service starting JBoss and deploying the test application from the *Lomboz J2EE View*.

Write the following URL in the browser and you should see the result page:

http://localhost:8080/web/servlet/AdapterHTTP?ACTION_NAME=Sommatore&primo=2&secondo=2&NEW_SESSION=TRUE

3 Service Publications as Webservice

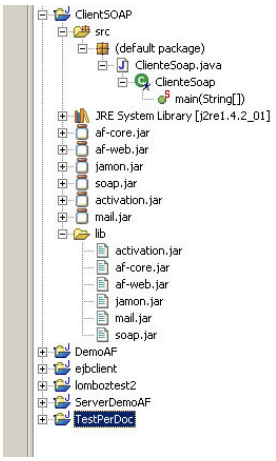
To use the service through a SOAP client you don't need to change the application configuration because Spago makes automatically all the services accessible by SOAP. You have only to check the correct behavior of the SOAP server using the following steps:

1. In the folder /conf/soap you should find the files soap.ds and soap.xml. The soap.ds file is generated by the Apache Soap implementation.
2. Copy in the folder /web the administration application of the Apache SOAP server (/admin).
3. Use the URL http://localhost:8080/web/admin/index.html to verify that the service is configured.
4. If the service is not configured you can activate it using the administration page, using the following information:

Deployed Service Information

'http://tempuri.org/it.eng.spago.dispatching.soapchannel.AdapterSOAP' Service Deployment Descriptor	
Property	**Details**
ID	http://tempuri.org/it.eng.spago.dispatching.soapchannel.AdapterSOAP
Scope	Session
Provider Type	Java
Provider Class	it.eng.spago.dispatching.soapchannel.AdapterSOAP
Use Static Class	False
Methods	service
Type Mappings	
Default Mapping Registry Class	

To execute the service you need a Java project containing a client using the following libraries:



- activation.jar
- spago-core.jar
- spago-web.jar
- mail.jar
- soap.jar

The services can be invoked by this client:

```
import java.net.URL;

import it.eng.spago.dispatching.soapchannel.AdapterSOAPProxy;

/**
 * @author bernabei
 */
public class ClienteSoap {

	public static void main(String[] args) {
		try {
			AdapterSOAPProxy adapterSOAP = new AdapterSOAPProxy();
			String request =
				"<SERVICE_REQUEST ACTION_NAME=\"Sommatore\" primo=\"2\" secondo=\"2\"
NEW_SESSION=\"TRUE\"/>";
			System.out.println("AdapterSOAPClient::main: request\n" + request);
			URL url=new URL("http://localhost:8080/web/servlet/rpcrouter");
			adapterSOAP.setEndPoint(url);
			String response = adapterSOAP.service(request);
			System.out.println(
				"AdapterSOAPClient::main: response\n" + response);
```

```
            } // try
            catch (Exception ex) {
                  System.out.println(
                        "AdapterSOAPClient::main: errore in adapterSOAPStub.service(request)");
                  ex.printStackTrace();
            } // catch (Exception ex) try
      } // public static void main(String[] args)
}
```

The response should be:

```
<RESPONSE>
  <SERVICE_RESPONSE risultato="4"/>
  <ERRORS/>
</RESPONSE>
```