Rampart y Axis2

Índice

Axis2, en su distribución básica, no ofrece la posibilidad de utilizar seguridad en los servicios web. Para tal fin hay que descargar el módulo de Rampart para Axis2. Descargamos la versión 1.5 de la web oficial del proyecto, http://ws.apache.org/rampart/. Descomprimimos en alguna ruta y copiamos los módulos y las librerías (bibliotecas) en sus carpetas correspondientes de Axis2:

```
servicios@servicios:~$ cp rampart-1.5/modules/*
~/axis2-1.5.1/repository/modules/
servicios@servicios:~$ cp rampart-1.5/lib/* ~/axis2-1.5.1/lib/
```

(Re)iniciamos Axis2. Para terminar una ejecución del servidor hay que pulsar Ctrl-C.

```
axis2-1.5.1/bin$ ./axis2server.sh
```

1. Cliente seguro

Podemos, o bien crear un nuevo proyecto y añadirle la gestión de dependencias de Maven, creando antes las carpetas correspondientes, o bien copiar un proyecto existente. Vamos a copiar el proyecto `SWSeguroCliente`, y pegarlo como `Axis2SWSeguroCliente`. Aprovecharemos el mismo documento WSDL. Sin embargo, vamos a eliminar el archvio `Seguro_SeguroSOAP_Client.java` del paquete custom, y también vamos a eliminar todo el paquete `es.ua.jtech.seguro`, para volver a generarlo. La instrucción de generación también cambiará, porque es diferente en Axis2. Editamos `GeneraCodigo.java`, que tiene que quedar así:

```
package es.ua.jtech.servcweb.hola.generador;

import org.apache.axis2.wsdl.WSDL2Code;

public class GeneraCodigo {
	public static void main(String[] args) throws Exception{
		WSDL2Code.main(new String[]{"-uw",
				"-S", "src/main/java",
				"-uri",
"src/main/resources/Seguro.wsdl"});

		System.out.println("Generado, no olvide hacer
Refresh.");
	}

}
```

Este código dará un error de compilación (no encontrará el import indicado) si no añadimos a las dependencias del proyecto la generación de código para Axis2. Editamos el `pom.xml` y añadimos las siguientes dependencias:

- axis2-codegen
- axis2-adb-codegen

- axis2-transport-http
- axis2-transport-local

Todas ellas deben pertenecer al grupo `org.apache.axis2`, y estar en su versión 1.5.1 (o posterior). Antes de guardarlo, aprovechamos para eliminar la dependencia de CXF, y también podemos eliminar el repositorio `apache-incubating`. Finalmente el `pom.xml` queda así:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
              http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>SWSeguroClient</groupId>
  <artifactId>SWSeguroClient</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
        <dependency>
                <groupId>xerces</groupId>
                <artifactId>xercesImpl</artifactId>
                <version>2.9.1</version>
        </dependency>
        <dependency>
                <groupId>org.apache.rampart</groupId>
                <artifactId>rampart-core</artifactId>
                <version>1.5</version>
        </dependency>
        <dependency>
                <groupId>org.apache.rampart</groupId>
                <artifactId>rampart-policy</artifactId>
                <version>1.5</version>
        </dependency>
        <dependency>
                <groupId>org.apache.axis2</groupId>
                <artifactId>axis2-codegen</artifactId>
                <version>1.5.1</version>
        </dependency>
        <dependency>
                <groupId>org.apache.axis2</groupId>
                <artifactId>axis2-adb-codegen</artifactId>
                <version>1.5.1</version>
        </dependency>
        <dependency>
                <groupId>org.apache.axis2</groupId>
                <artifactId>axis2-transport-http</artifactId>
                <version>1.5.1</version>
        </dependency>
        <dependency>
                <groupId>org.apache.axis2</groupId>
                <artifactId>axis2-transport-local</artifactId>
                <version>1.5.1</version>
        </dependency>
  </dependencies>
  <build>
          <plugins>
                  <plugin>
```

```
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
                        <configuration>
                                <source>1.6</source>
                                <target>1.6</target>
                        </configuration>
                </plugin>
        </plugins>
  </build>
</project>
```

Lo guardamos y esperamos a que se descarguen las dependencias y se construya el espacio de trabajo. Si todo ha ido bien, el error de `GeneraCodigo.java` habrá desaparecido. Antes de ejecutarlo para generar el código, vamos a modificar el WSDL para pasar a una versión anterior de las política, y de la política de seguridad.

Editamos `src/main/resources/Seguro.wsdl`. En la etiqueta `wsdl:definitions` cambiamos los espacios de nombres `xmlns:sp` y `xmlns:wsp` a:

```
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
```

También modificamos los `sp:IncludeToken`. Todos ellos (tres ocurrencias, en nuestro documento WSDL) hacen referencia al estándar

```
http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/*
```

donde (*) es `AlwaysToRecipient`, o `Never`. Debemos cambiarlas al estándar

```
http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/*
```

Lo guardamos y ya podemos generar el código y refrescar. Ahora tenemos que crear una clase con un método main. En el paquete `es.ua.jtech.seguro.custom` añadimos una nueva clase llamada `Cliente` y marcamos la casilla del método main en el asistente. Introducimos el siguiente código:

```
package es.ua.jtech.seguro.custom;

import java.io.FileNotFoundException;
import java.rmi.RemoteException;

import javax.xml.stream.XMLStreamException;

import org.apache.axiom.om.impl.builder.StAXOMBuilder;
import org.apache.axis2.context.ConfigurationContext;
import org.apache.axis2.context.ConfigurationContextFactory;
import org.apache.neethi.Policy;
import org.apache.neethi.PolicyEngine;

import es.ua.jtech.seguro.SeguroStub;
```

```
public class Cliente {

        public static void main(String[] args)
          throws RemoteException, FileNotFoundException,
XMLStreamException {
                ConfigurationContext context =
ConfigurationContextFactory.
          reateConfigurationContextFromFileSystem("repository");
                SeguroStub servicio = new SeguroStub(context,
                "http://localhost:1234/axis2/services/Seguro");
servicio._getServiceClient().engageModule("rampart");
                StAXOMBuilder builder =
                  new
StAXOMBuilder("src/main/resources/rampartConfig.xml");
                Policy rampartPolicy =
PolicyEngine.getPolicy(builder.getDocumentElement());
servicio._getServiceClient().getAxisService().getPolicySubject().
                  attachPolicy(rampartPolicy);
                System.out.println(servicio.saluda("Boyan",
"Bonev"));
        }

}
```

En el main se crea el contexto de configuración a partir de un "repository" que debemos añadir a nuestro proyecto cliente. Creamos en la base del proyecto la carpeta `repository/modules`. Copiamos dentro de modules los archivos

```
rampart-1.5/modules/rahas-1.5.mar
rampart-1.5/modules/rampart-1.5.mar
```

Después de activar el uso del módulo rampart, el cliente crea la configuración de rampart a partir del archivo

```
src/main/resources/rampartConfig.xml
```

Vamos a crearlo e introducir en él la información del nombre de usuario, el alias para el certificado de firma y de encriptación, y la configuración de criptografía de la firma y de la encriptación, que va a contener la misma información:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
        xmlns="http://ws.apache.org/rampart/policy">
        <RampartConfig>
                <user>boyan</user>
                <userCertAlias>cliente1</userCertAlias>
                <encryptionUser>servidor1</encryptionUser>
                <passwordCallbackClass>
es.ua.jtech.seguro.custom.PasswordCallbackHandler
                </passwordCallbackClass>
                <signatureCrypto>
```

```
                        <crypto
provider="org.apache.ws.security.components.crypto.Merlin">
                                <property
name="org.apache.ws.security.crypto.merlin.keystore.
type">
                                        JKS
                                </property>
                                <property
name="org.apache.ws.security.crypto.merlin.file">
/home/servicios/claves/cliente.ks
                                </property>
                                <property
name="org.apache.ws.security.crypto.merlin.keystore.
password">
                                        cliente1-kspass
                                </property>
                        </crypto>
                </signatureCrypto>
                <encriptionCrypto>
                        <crypto
provider="org.apache.ws.security.components.crypto.Merlin">
                                <property
name="org.apache.ws.security.crypto.merlin.keystore.
type">
                                        JKS
                                </property>
                                <property
name="org.apache.ws.security.crypto.merlin.file">
/home/servicios/claves/cliente.ks
                                </property>
                                <property
name="org.apache.ws.security.crypto.merlin.keystore.
password">
                                        cliente1-kspass
                                </property>
                        </crypto>
                </encriptionCrypto>
        </RampartConfig>
</wsp:Policy>
```

Una vez creado este fichero, ya no necesitamos la información contenida en `crypto.properties`, y eliminamos este archivo.

Nótese que la contraseña del usuario `<user>` no se ha introducido en el fichero de configuración. Ésta va a ser introducida por el manejador de passwords, que, como indica la configuración que hemos introducido, debe encontrarse en:

```
es.ua.jtech.seguro.custom.PasswordCallbackHandler
```

Creamos esa clase con el asistente, pulsando sobre el paquete custom, New / Java Class, con el nombre `PasswordCallbackHandler`, y le añadimos la interfaz `CallbackHandler`, dejando marcada la casilla "Inherited abstract methods", que por defecto está marcada. Se generará la clase con la función handle, dentro de la cuál insertamos el siguiente código:

```

```

```
                for(Callback cb : callbacks){
                        WSPasswordCallback pcb =
(WSPasswordCallback)cb;
                        switch(pcb.getUsage()){
                        case WSPasswordCallback.SIGNATURE:
                        case WSPasswordCallback.DECRYPT:
if(pcb.getIdentifier().equals("cliente1")){
pcb.setPassword("cliente1-pass");
                                }
                                break;
                        case WSPasswordCallback.USERNAME_TOKEN:
if(pcb.getIdentifier().equals("boyan")){
pcb.setPassword("boyan-pass");
                                }
                        }
                }
```

Añadimos los imports que falten y guardamos. Si ejecutamos `es.ua.jtech.seguro.custom.Cliente`, con el TCP Monitor escuchando al puerto 1234 y reenviando al 8080, donde debe estar escuchando el servidor de Axis2, debemos capturar en el TCP Monitor el mensaje encriptado y firmado por el cliente. La respuesta por parte del servidor sería una excepción de que no encuentra el servicio en el endpoint. Vamos a implementarlo a continuación.

2. Servidor seguro

Copiamos el proyecto cliente, `Axis2SWSeguroCliente`, y lo pegamos con el nombre `Axis2SWSeguroServidor`. El archivo `pom.xml` se puede quedar igual, aunque podemos cambiar el Group Id y el Artifact Id a `SWSeguroServidor`. Eliminamos la carpeta `repository`, que no va a hacer falta, ya que este proyecto se empaquetará y se desplegará en el servidor de Axis2, que ya cuenta con su propia carpeta `repository`, en la cual, además, ya habíamos copiado los módulos de rampart. Eliminamos también el paquete `es.ua.jtech.seguro`, y dentro del paquete `es.ua.jtech.seguro.custom` eliminamos `Cliente.java`, pero dejamos el `PasswordCallbackHandler`, ya que Rampart lo utilizará. Editamos `GeneraCodigo.java` para introducir los parámetros de generación del código de servidor, e indicar la ruta `META-INF` que se creará en la carpeta de recursos. Ahí se copiará el WSDL y se generará un archvio `services.xml`, que después editaremos para incluir la configuración de Rampart. En `GeneraCodigo.java` cambiamos los parámetros a:

```
                WSDL2Code.main(new String[]{"-ss",
                                "-sd", "-uw",
                                "-S", "src/main/java",
                                "-R",
"src/main/resources/META-INF",
                                "-uri",
"src/main/resources/Seguro.wsdl"});
```

Lo ejecutamos y pulsamos Refresh sobre el proyecto. Aparte de las clases del paquete `es.ua.jtech.seguro`, también nos encontramos con la nueva carpeta `src/main/resources/META-INF`. Vamos a editar el archivo `services.xml` para indicarle que use rampart. Dentro de la sección `<service>` introducimos:

```
                <module ref="rampart" />
                <wsp:Policy
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd"
                        wsu:Id="p1">

                        <RampartConfig
xmlns="http://ws.apache.org/rampart/policy">
                                <!-- Pegar aquí el contenido de la
RampartConfig de
rampartConfig.xml -->
                                <!-- Cambiar el keystore a
servidor.ks y la password
a servidor1-kspass -->

                                <!-- Eliminar <user> y cambiar los
siguientes alias     -->
<userCertAlias>servidor1</userCertAlias>
<encryptionUser>useReqSigCert</encryptionUser>


                        </RampartConfig>

                        <!-- Pegar aquí el contenido de la Policy
de Seguro.wsdl, incluído
                        el AsymmetricBinding, Wss10, SignedParts,
EncryptedParts,
SignedSupportingTokens -->

                </wsp:Policy>
```

Como está indicado en los comentarios, podemos copiar y pegar la configuración que tenemos en `rampartConfig.xml`, y a continuación podemos eliminar este archivo porque no nos va a hacer falta en el proyecto del servidor. No debemos olvidar cambiar los nombres de los keystore, al del keystore del servidor, y la contraseña correspondiente. Asimismo vamos a cambiar el alias del certificado usado para firmar, ya que va a ser el servidor quien firme con su clave pública (recordemos que en el keystore del servidor no está el certificado del cliente). En cuanto a la encriptación, el servidor encriptará con el certificado cliente, que el cliente incluye en sus mensajes, y eso se le indica a rampart con el nombre `useReqSigCert`. Eliminamos el nombre de usuario, ya que el servidor no envía información de login al cliente, en este caso.

Una vez hechos todos estos cambios en la configuración de rampart, hay que actualizar el manejador de contraseñas del servidor, que habíamos reutilizado del cliente. Hay que

cambiar el caso `SIGNATURE | DECRYPT` para que asigne la contraseña "`servidor1-pass`" cuando el identificador sea "`servidor1`". El caso de `USERNAME_TOKEN` lo podemos eliminar, ya que el servidor no se autenticará a nivel de login de usuario contra el cliente. Tras los cambios, la `es.ua.jtech.seguro.custom.PasswordCallbackHandler` queda así:

```
package es.ua.jtech.seguro.custom;

import java.io.IOException;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import org.apache.ws.security.WSPasswordCallback;

public class PasswordCallbackHandler implements CallbackHandler {

        @Override
        public void handle(Callback[] callbacks) throws
IOException,
                        UnsupportedCallbackException {
                for(Callback cb : callbacks){
                        WSPasswordCallback pcb =
(WSPasswordCallback)cb;
                        switch(pcb.getUsage()){
                        case WSPasswordCallback.SIGNATURE:
                        case WSPasswordCallback.DECRYPT:
if(pcb.getIdentifier().equals("servidor1")){
pcb.setPassword("servidor1-pass");
                                }
                                break;
                        }
                }
        }

}
```

Con esto ya tenemos configurada la seguridad. Sólo falta implementar la operación del servicio, en el Skeleton correspondiente. Ya que vamos a editarlo, lo movemos al paquete `custom`. Pulsamos con el botón derecho sobre `SeguroSkeleton.java` del paquete `es.ua.jtech.seguro` y con la opción Refactor / Move... lo cambiamos al paquete `es.ua.jtech.seguro.custom`. Antes de que se nos olvide, vamos a editar de nuevo el archivo `services.xml`, donde la línea

```
<parameter
name="ServiceClass">es.ua.jtech.seguro.SeguroSkeleton</parameter>
```

indica el paquete antiguo del `SeguroSkeleton`. La cambiamos a:

```
<parameter
name="ServiceClass">es.ua.jtech.seguro.custom.SeguroSkeleton
</parameter>
```

Ahora podemos seguir editando `SeguroSkeleton.java`. Copiamos el contenido de la función saluda de la clase `es.ua.jtech.seguro.custom.SegurImpl.java` que teníamos en el proyecto de CXF llamado `SWSeguroServicio`. También copiamos la función `loggedIn( )` a la que se realiza una llamada. Dará un error porque ya no tenemos la variable `wsContext`, cuyo valor obteníamos antes por inyección, gracias a una anotación de Java. Así que vamos a modificar la primera línea de la función `loggedIn( )`, para obtener el contexto del mensaje con el método estático `getCurrentMessageContext` de la clase `org.apache.axis2.context.MessageContext` de las librerías de Axis2:

```
        Vector<WSHandlerResult> handlerResults =
(Vector<WSHandlerResult>)org.apache.axis2.context.MessageContext.
getCurrentMessageContext().getProperty(WSHandlerConstants.RECV_RESULTS);
```

Podemos ignorar la advertencia de Type Safety, o bien añadir la etiqueta `@SupressWarnings` al principio de la función. Ahora la clase no debería tener ningún error. El código final es:

```
package es.ua.jtech.seguro.custom;

import java.util.Vector;

import org.apache.ws.security.WSConstants;
import org.apache.ws.security.WSSecurityEngineResult;
import org.apache.ws.security.WSUsernameTokenPrincipal;
import org.apache.ws.security.handler.WSHandlerConstants;
import org.apache.ws.security.handler.WSHandlerResult;

/**
 * SeguroSkeleton java skeleton for the axisService
 */
public class SeguroSkeleton {

        public String saluda(String nombre, String apellido) {
                if (loggedIn()) {
                        return "¡Hola, " + nombre.substring(0, 1) +
". " + apellido + "!";
                } else {
                        return "Login incorrecto, le retiro el
saludo.";
                }
        }

        @SuppressWarnings("unchecked")
        private boolean loggedIn() {
                Vector<WSHandlerResult> handlerResults =
                      (Vector<WSHandlerResult>)
org.apache.axis2.context.MessageContext
```

```
                        .getCurrentMessageContext().getProperty(
WSHandlerConstants.RECV_RESULTS);
                for (WSHandlerResult handlerResult :
handlerResults) {
                        for (WSSecurityEngineResult handler :
                             (Vector<WSSecurityEngineResult>)
handlerResult.getResults()) {
                                int action = ((Integer) handler
.get(WSSecurityEngineResult.TAG_ACTION)).intValue();
                                if (action == WSConstants.UT) {
                                        WSUsernameTokenPrincipal
utp =
(WSUsernameTokenPrincipal) handler
.get(WSSecurityEngineResult.TAG_PRINCIPAL);
                                        if (utp != null) {
                                                if
(utp.getName().equals("boyan")
                                                                &&
utp.getPassword().equals("boyan-pass")) {
                                                        return
true;
                                                }
                                        }
                                }
                        }
                }
                return false;
        }

}
```

También, después de todas las modificaciones, el XML final del archivo `services.xml` sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
        <!-- This file was auto-generated from WSDL -->
        <!--
                by the Apache Axis2 version: 1.5.1 Built on : Oct
19, 2009 (10:59:00
                EDT)
        -->
<serviceGroup>
        <service name="Seguro">
                <messageReceivers>
                        <messageReceiver
mep="http://www.w3.org/ns/wsdl/in-out"
class="es.ua.jtech.seguro.SeguroMessageReceiverInOut" />
                </messageReceivers>
                <parameter
name="ServiceClass">es.ua.jtech.seguro.custom.SeguroSkeleton
                </parameter>
                <parameter name="useOriginalwsdl">true</parameter>
                <parameter
name="modifyUserWSDLPortAddress">true</parameter>
                <operation name="saluda"
mep="http://www.w3.org/ns/wsdl/in-out"
```

```
                        namespace="http://jtech.ua.es/Seguro/">
<actionMapping>http://jtech.ua.es/Seguro/saluda</actionMapping>
<outputActionMapping>http://jtech.ua.es/Seguro/Seguro/saludaResponse
                        </outputActionMapping>
                </operation>
                <module ref="rampart" />
                <wsp:Policy
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd"
                        wsu:Id="p1">

                        <RampartConfig
xmlns="http://ws.apache.org/rampart/policy">
<userCertAlias>servidor1</userCertAlias>
<encryptionUser>useReqSigCert</encryptionUser>
                                <passwordCallbackClass>
es.ua.jtech.seguro.custom.PasswordCallbackHandler
                                </passwordCallbackClass>
                                <signatureCrypto>
                                        <crypto
provider="org.apache.ws.security.components.crypto.Merlin">
                                                <property
name="org.apache.ws.security.crypto.merlin.
keystore.type">JKS</property>
                                                <property
name="org.apache.ws.security.crypto.merlin.
file">/home/servicios/claves/servidor.ks</property>
                                                <property
name="org.apache.ws.security.crypto.merlin.
keystore.password">servidor1-kspass</property>
                                        </crypto>
                                </signatureCrypto>
                                <encriptionCrypto>
                                        <crypto
provider="org.apache.ws.security.components.crypto.Merlin">
                                                <property
name="org.apache.ws.security.crypto.merlin.
keystore.type">JKS</property>
                                                <property
name="org.apache.ws.security.crypto.merlin.
file">/home/servicios/claves/servidor.ks</property>
                                                <property
name="org.apache.ws.security.crypto.merlin.
keystore.password">servidor1-kspass</property>
                                        </crypto>
                                </encriptionCrypto>

                        </RampartConfig>

                        <sp:AsymmetricBinding>
                                <wsp:Policy>
                                        <sp:InitiatorToken>
                                                <wsp:Policy>
<sp:X509Token
sp:IncludeToken="http://schemas.xmlsoap.org/ws/
2005/07/securitypolicy/IncludeToken/
AlwaysToRecipient">
```

```
<wsp:Policy>
<sp:WssX509V3Token10 />
</wsp:Policy>
</sp:X509Token>
                                                        </wsp:Policy>
                                                </sp:InitiatorToken>
                                                <sp:RecipientToken>
                                                        <wsp:Policy>
<sp:X509Token
sp:IncludeToken="http://schemas.xmlsoap.org/ws/
2005/07/securitypolicy/IncludeToken/Never">
<wsp:Policy>
<sp:WssX509V3Token10 />
</wsp:Policy>
</sp:X509Token>
                                                        </wsp:Policy>
                                                </sp:RecipientToken>
                                                <sp:AlgorithmSuite>
                                                        <wsp:Policy>
<sp:TripleDesRsa15 />
                                                        </wsp:Policy>
                                                </sp:AlgorithmSuite>
                                                <sp:EncryptBeforeSigning />
                                        </wsp:Policy>
                                </sp:AsymmetricBinding>
                                <sp:Wss10>
                                        <wsp:Policy>
<sp:MustSupportRefEmbeddedToken />
<sp:MustSupportRefIssuerSerial />
                                        </wsp:Policy>
                                </sp:Wss10>
                                <sp:SignedParts>
                                        <sp:Body />
                                </sp:SignedParts>
                                <sp:EncryptedParts>
                                        <sp:Body />
                                </sp:EncryptedParts>
                                <sp:SignedSupportingTokens>
                                        <wsp:Policy>
                                                <sp:UsernameToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/
securitypolicy/IncludeToken/AlwaysToRecipient" />
                                        </wsp:Policy>
                                </sp:SignedSupportingTokens>
                        </wsp:Policy>
                </service>
</serviceGroup>
```

3. Ejercicios

3.1. Calculadora con firma digital en Axis2

El enunciado del ejercicio "Calculadora con firma digital" es:

- Impleméntese un servicio de calculadora que realice la operación suma de dos números y la operación resta de dos números, devolviendo un único resultado de tipo double en cada una de ellas. El servicio deberá soportar firma digital, y el cliente deberá firmar los mensajes SOAP.

En este ejercicio debemos adaptar la calculadora con firma digital a Axis2, desplegar el servicio seguro en Axis2, y probar con el cliente que funciona correctamente.

Utilícese el TCP Monitor para observar el contenido de los mensajes y la firma que debe acompañarlos.