AGESIC

Gerencia de Proyectos

Tutorial para Consumir servicios de la PGE sobre Plataforma Java

Historial de Revisiones

Fecha	Versión	Descripción	Autor	Aprobado Por
08/11/2011	1.0	Versión inicial	Guzmán Llambías	Guzmán Llambías
16/12/2011	2.0	Mejoras en la forma de establecer la comunicación SSL.	Marcelo Caponi	Guzmán Llambías
05/06/2012	2.1	Correcciones menores	Guzmán Llambías	Guzmán Llambías
01/10/2012	2.2	Adaptación a la versión 1.5 del PGEClient.jar, e inclusión de WSSecurity	Sergio Pío Alvarez	
30/04/2013	3.0	Agregada la sección 1.4, Configuración del ambiente (antes era un documento aparte) Ajustes por cambios en el FTP	Sergio Pío Alvarez	
06/09/2013	3.1	Corrección del rol utilizando en la parte incremental. Corrección de la explicación de cómo generar el valor de wsa:action en el caso de que el WSDL no lo especifique.	Sergio Pío Alvarez	
12/08/14	3.2	Se cambia descripción de servicio y método invocado.	Sebastián Filippini	

Índice de contenido

1 Introducción

1.1 Objetivo

El objetivo de este tutorial es proveer una guía paso a paso para el desarrollo de un cliente stand-alone de la Plataforma de Gobierno Electrónico (PGE) sobre la plataforma Java para consumir un servicio web que ya se encuentra publicado, para lo cual se utilizará un ejemplo concreto.

1.2 Prerrequisitos

Se asume que el usuario conoce, a un nivel básico, las especificaciones WS-Security [1], WS-Trust [2] y SAML 1.1 [3]. Además, se asume que el usuario está familiarizado con el uso de certificados, keystores, aplicaciones JavaEE y servicios web.

1.3 Requerimientos del software

La tabla 1 presenta las herramientas y productos de *software* requeridos para desarrollar y ejecutar la Aplicación Cliente. Si bien pueden usarse otras herramientas, y obtener el mismo resultado, en este documento se asumirá el uso de las mencionadas en la tabla.

Producto	Versión
Java Developer Kit (JDK)	6.0
JBoss Application Server	5.1
JBoss Web Services	3.2.2.GA
Eclipse	3.5 /Galileo
JBossWS Tools	3.1 GA
OpenSAML	2.3.1

Tabla 1 – Requerimientos de Software

1.4 Preparación del ambiente

A continuación se describe paso a paso la preparación del ambiente para el desarrollo del tutorial Java. Se asume que el lector ya ha descargado e instalado en su equipo el JRE, el entorno de desarrollo Eclipse y el servidor de aplicaciones JBoss AS según fue detallado en los requerimientos en la sección anterior.

La preparación del ambiente incluye las siguientes etapas:

1. Remplazar bibliotecas nativas de Java.
2. Definir el JBoss AS Runtime.
3. Definir el JBossWS Runtime.

1.4.1 Sobreescribir bibliotecas nativas de Java

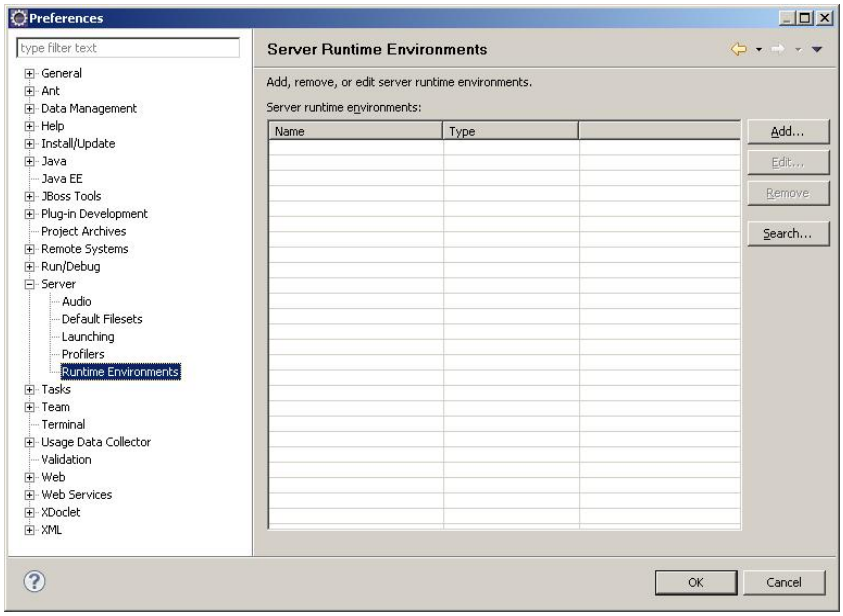
La versión 6 de Java provee una implementación nativa para Web Services que no es compatible con la tecnología JBoss utilizada para desarrollar el cliente en este tutorial. Por lo tanto, es necesario remplazar esta biblioteca utilizando los mecanismos tradicionales que provee Java. Para ello, se debe copiar los archivos de la carpeta endorsed del archivo de materiales descargado a la carpeta <JRE_HOME>\lib\endorsed. En caso de no existir dicha carpeta se debe crearla crearla.

Nota: <JRE_HOME> debe ser remplazado por la ruta completa a la carpeta donde está instalada el Java Runtime Environment (JRE).

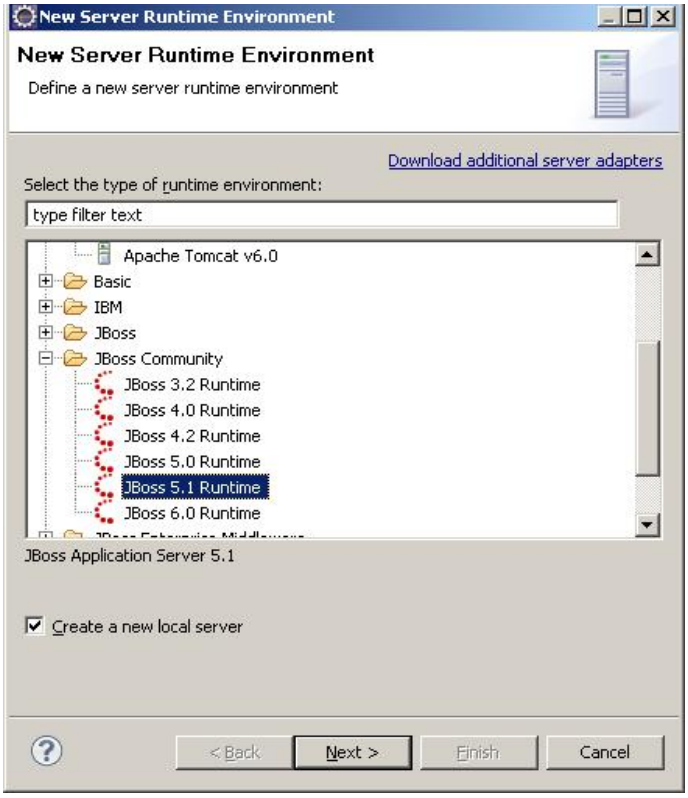
1.4.2 Definir el JBoss AS Runtime

El segundo paso consiste en registrar el servidor de aplicaciones JBoss AS en el entorno de desarrollo Eclipse, de forma de poder crear aplicaciones asociadas a dicho servidor. Para ello, se debe seguir los siguientes pasos:

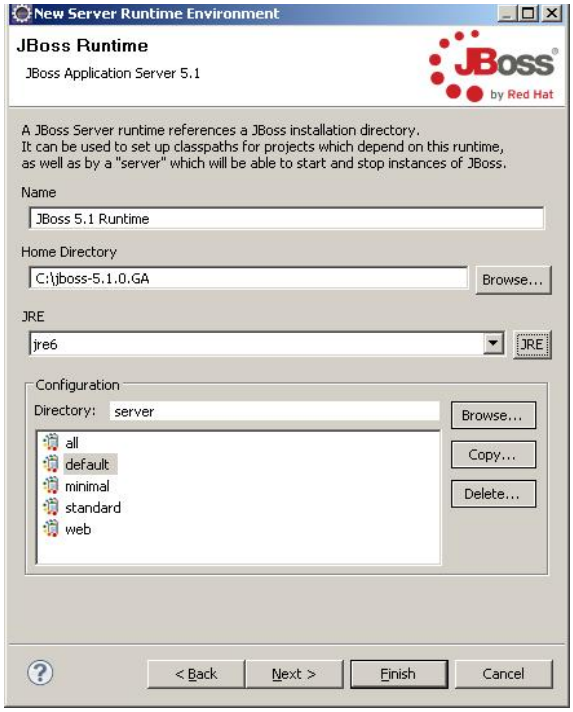
1. Seleccionar del menú de Eclipse la opción Windows → Preferences.
2. Seleccionar la opción Server → Runtime Environments como se muestra en la siguiente imagen:



3. Hacer clic en el botón Add..., luego la opción JBoss Community → JBoss 5.1 Runtime como se muestra en la figura siguiente y luego hacer clic en el botón Next.



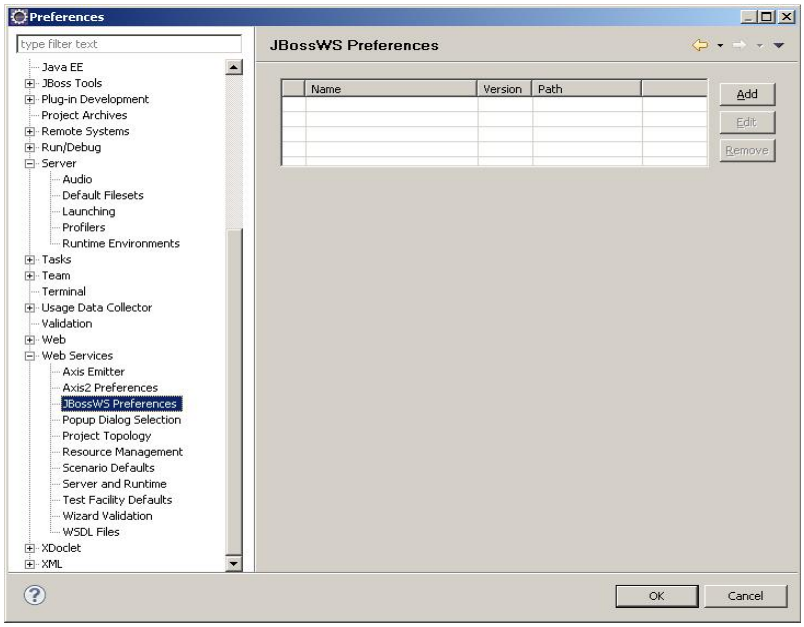
4. Completar los datos solicitados: en el campo Name ingresar un nombre descriptivo del servidor, por ejemplo, JBoss Tutorial; en el campo Home Directory ingresar la ruta completa al directorio donde se encuentra instalado el servidor de aplicaciones JBoss AS; en el campo JRE seleccionar la Java Runtime Environment deseada (debe ser 1.6 o superior para este tutorial) y luego en el campo configuración seleccionar la configuración "default". Finalmente, seleccionar y presionar el botón Finish. Se debe alcanzar un resultado similar al de la figura 4.



1.4.3 Definir el JBossWS Runtime

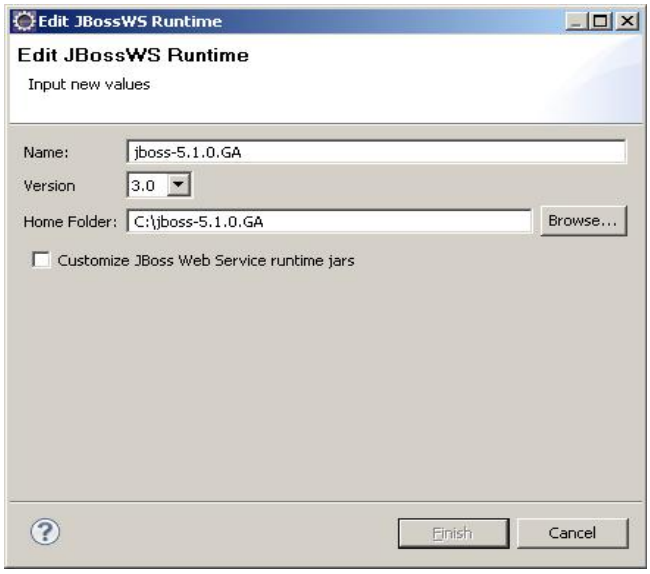
El tercer y último paso es registrar el JBoss WS Runtime en el entorno de desarrollo Eclipse. Para esto, se deben completar los siguientes pasos:

1. Seleccionar del menú de Eclipse la opción Windows → Preferences.
2. En el menú izquierdo, seleccionar Web Services → JBossWS Preferences y presionar el botón Add.. .



3. Completar los datos solicitados: en el campo Name ingresar un nombre descriptivo, en el campo Version seleccionar 3.0 y en el campo Home Folder especificar la ruta completa a la instalación del servidor de aplicaciones JBoss AS. Dejar el campo

Customize JBoss Web Service runtime jars sin marcar. Finalmente hacer clic en el botón Finish.



2 Descripción del escenario

La figura 1 presenta el escenario de ejemplo que se utiliza en este tutorial, en el cual intervienen dos organismos: el Banco de Previsión Social (BPS) que será el Organismo Cliente (quien consume el servicio) y AGESIC que será el Organismo Proveedor.

AGESIC provee el servicio "**Timestamp**" el cual tiene una unica operaciòn "**GetTimestamp**". Cuando se registró el servicio en la PGE, se creó un Servicio Proxy para que las Aplicaciones Cliente accedan al servicio a través de él (los clientes se comunican con el proxy y éste transfiere la invocación al servicio final; luego toma la respuesta de este último y la reenvía al cliente). Además, mediante la configuración de políticas de control de acceso, el AGESIC autorizó a los usuarios con rol "gerencia de proyectos" de la sección "agesic" (ou=gerencia de proyectos,o=agesic) a consumir el método "GetTimestamp" [1].

Por otro lado, en el BPS hay una Aplicación Cliente que está siendo utilizada por el usuario Pruebas que tiene el rol mencionado. La aplicación necesita acceder al servicio de AGESIC para lo cual, utilizando las credenciales del usuario Pruebas y a través de una Aplicación Emisora de Tokens interna al BPS, obtiene un *token* de seguridad SAML firmado por el BPS (pasos 1.a y 1.b).

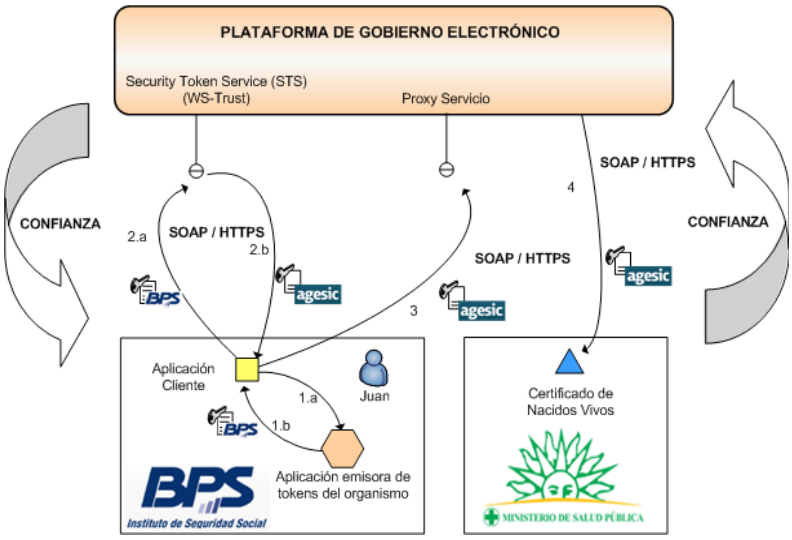


Figura *1*: Escenario de uso

Luego con el *token* recibido obtiene del STS de la PGE, utilizando el estándar WS-Trust, otro *token* de seguridad firmado por la plataforma (pasos 2.a y 2.b). Para emitir este *token* la PGE verifica la firma digital del *token* enviado por la aplicación y la existencia del rol "ou=gerencia de proyectos,o=agesic".

Por último, la Aplicación Cliente invoca al Servicio del MSP a través del Servicio Proxy de la PGE (los clientes nunca acceden al servicio final directamente, siempre lo hacen a través del proxy creado en la PGE; existe un proxy específico para cada servicio disponible a través de la PGE). En la invocación se incluye el *token* firmado por la PGE y se especifican el servicio y el método a invocar.

La tabla 2 especifica algunos de los datos a utilizar en la implementación del

[1] *Los roles autorizados a invocar una determinada operación de un servicio web son acordados entre el proveedor del servicio y AGESIC. Los clientes que deseen invocar cada operación deberán solicitar esta información a AGESIC.*

escenario.

Dato	Valor	Comentarios
Nombre de Usuario	Pruebas	Este dato es solo con fines de auditoría de la PGE.
Rol de Usuario	ou=gerencia de proyectos,o=agesic	Este dato debe ser solicitado por el organismo cliente a AGESIC. Es propio del organismo e incluso puede ser diferente para cada proyecto dentro del organismo.
Dirección Lógica del Servicio	`http://testservicios.pge.red.uy/timestamp`	Este dato será proporcionado por AGESIC. Es una URI que permite a la PGE identificar el servicio que se desea invocar. Se corresponde con wsa:To.
Método del Servicio	http://www.agesic.gub.uy/soa/TimestampService/TimestampServiceImplPort/GetTimestamp	Debe especificarse un valor para ser enviado en el capo wsa:Action. Para determinar el valor que hay que especificar, ver el apéndice 4.
PolicyName[2]	urn:tokensimple	En producción, es urn:std15.

Tabla 2 – Datos para la Implementación del Escenario

Los datos de negocio a incluir en la invocación, están especificados en la descripción del servicio (WSDL). En esta descripción también se incluye la URL del Servicio Proxy donde el cliente debe enviar los mensajes SOAP para invocar al servicio (valor del atributo location del tag soap:address, dentro del tag wsdl:service); en testing, esta URL debe comenzar con "https://testservicios.pge.red.uy" (la URL de la PGE).

[2] *Es la política de autenticación utilizada por AGESIC para la verificación de solicitudes del cliente. En el ambiente de Testing, el único valor aceptado es "urn:tokensimple"; en el ambiente de Producción, el único valor aceptado es urn:std15.*

3 Implementación del escenario

En esta sección se describe, paso a paso, la implementación de una Aplicación Cliente Java de escritorio según el escenario descrito previamente.

La implementación del escenario comprende las siguientes etapas:

1. Obtener los materiales necesarios: librerías, certificados, wsdl, etc.
2. Crear proyecto Java Faceted
3. Obtención del *token* de Seguridad emitido por la PGE
4. Invocación del Servicio

En las siguientes subsecciones se describe en detalle cada una de ellas.

3.1 Descargar los materiales necesarios

Como primer paso se deben descargar los materiales necesarios. Esto incluye las librerías adicionales que deberán ser agregadas a los clientes, los certificados digitales, y el WSDL que define el servicio que se invocará. Estos materiales se pueden obtener desde el FTP público de AGESIC, a través de la URL ftp://ftp.agesic.gub.uy/Tutoriales/Java/materiales.zip (nombre de usuario: **agesic**, contraseña: **publico**). Descargar este archivo a la ubicación que se desee y descomprimirlo.

3.2 Crear proyecto Java Faceted

1. Seleccionar *File → New → Other → General → Faceted Project,* crear un nuevo proyecto con el nombre *Tutorial_PGE* y los facets *Java **6.0***, *JBoss Web Service Core **3.0*** y *Dynamic Web Module **2.4*** según las figuras 2 y 3.

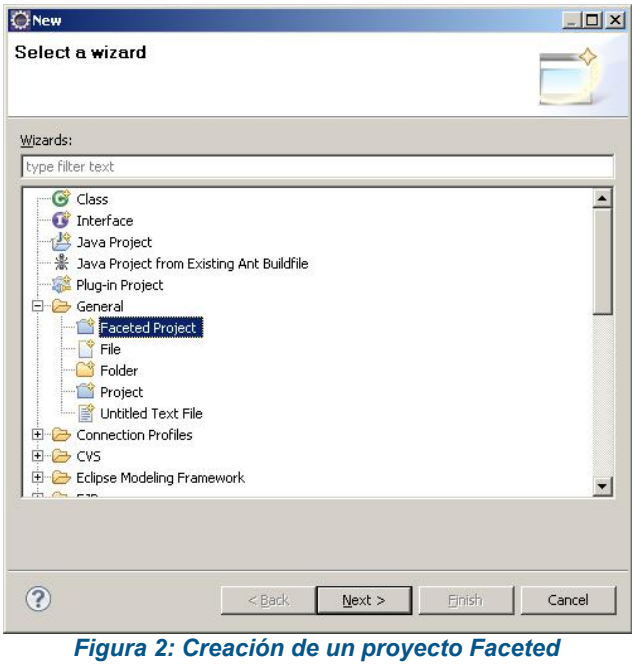


Figura 2: Creación de un proyecto Faceted

Nota: La aplicación Java que se está desarrollando no es una aplicación Web. Sin embargo, JBossWS Tools requiere que se utilice el faceted Web.

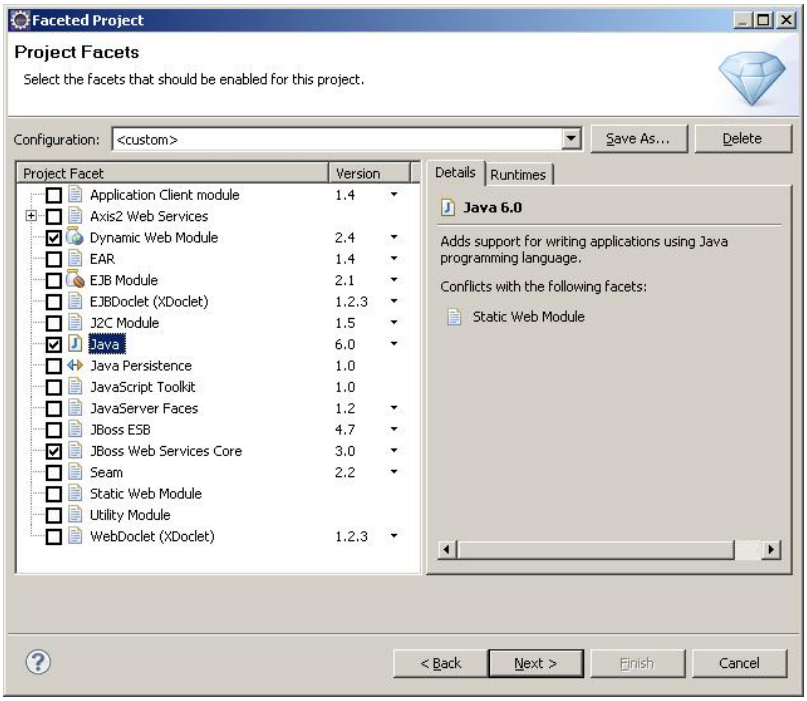


Figura 3: Selección de los facets

2. Configurar la carpeta destino del código fuente (src) y compilado (build), así como también el directorio de contenido Web.

3. Seleccionar el JBossWS Runtime como se ilustra en la figura 4 y presionar el botón Finish.

Figura 4: Configuración del JBossWS Runtime del proyecto

3.3 Incluir Librerías y Otros Archivos Necesarios

La Aplicación Cliente requiere las librerías de JBossWS y OpenSAML, así como la Librería PGEClient.jar implementada por AGESIC (versión 1.5 o posterior). A su vez, es necesario incluir el WSDL del servicio Timestamp. Para ello, se deben seguir los siguientes pasos:

1. Hacer clic derecho en el proyecto, seleccionar *New → Folder* y crear una carpeta llamada *lib*. Copiar en dicha carpeta **todos** los archivos que se encuentran en las carpetas lib/agesic, lib/http-components, lib/jbossws y lib/saml del archivo obtenido del FTP de AGESIC.

2. Agregar todas las bibliotecas copiadas en el paso anterior al Java Build Path del proyecto, haciendo clic derecho sobre el proyecto y luego seleccionado *Properties → Java Build Path → Libraries → Add JARs…*

3. Colocar la biblioteca JBossWS Runtime en el último lugar del classpath. Para ello, seleccionar la solapa *Order and Export,* seleccionar la biblioteca JBossWS Runtime y presionar el botón *Bottom* como se muestra en la figura 5.

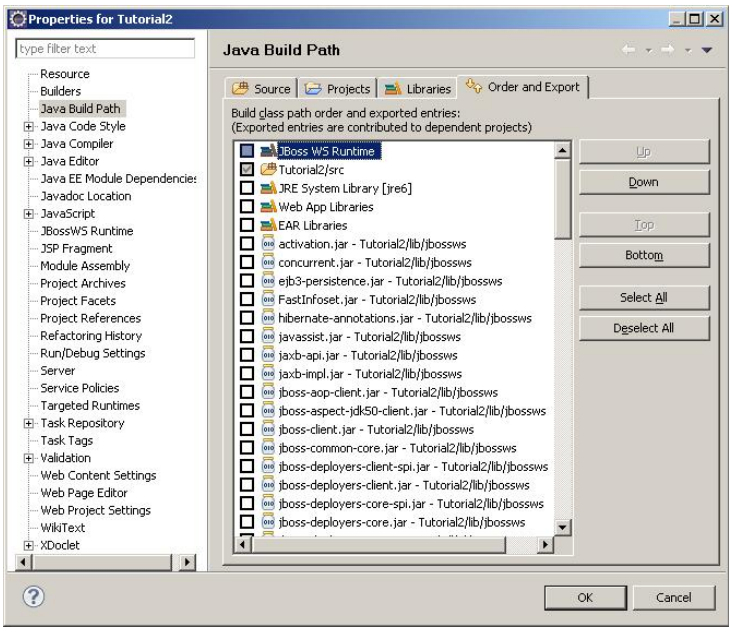


Figura 5 – Clase PGEClientTest

4. Crear una carpeta denominada wsdl y agregar todos los archivos de la carpeta wsdl del archivo obtenido del FTP de AGESIC.

5. Crear una carpeta denominada keystores y copiar todos los archivos de la carpeta keystores del archivo obtenido del FTP de AGESIC.

3.4 Obtención del token de Seguridad emitido por la PGE

Para realizar esta tarea, se utiliza el adaptador PGEClient.jar desarrollado por AGESIC. Los pasos a seguir son los siguientes:

1. Crear el package *test*. Para ello, seleccionar en el proyecto y luego clic derecho → new → package.

2. Crear la clase *PGEClientTest* en el package *test* de forma tal que contenga un método public static void *main(String[] args)* como se presenta en la figura 6.

```
package test;

public class PGEClientTest {
   public static void main(String[] args){
      //Aqui se pondra el codigo para invocar el servicio
   }
```

Figura 6 – Clase PGEClientTest

3. Importar las clases a utilizar como se muestra en la figura 7.

```
package test;

import uy.gub.agesic.beans.RSTBean;
import uy.gub.agesic.beans.SAMLAssertion;
import uy.gub.agesic.beans.StoreBean;
import uy.gub.agesic.exceptions.RequestSecurityTokenException;
import uy.gub.agesic.sts.client.PGEClient;

public class PGEClientTest {
   public static void main(String[] args){
      //Aqui se pondra el codigo para invocar el servicio
   }
```

Figura 7 – Importar las clases requeridas

4. Colocar en el método *main* el código que se muestra en la figura 8. Este código crea un RSTBean especificando los datos para enviar el pedido al STS de la PGE. En el pedido se carga la información relativa al usuario, organismo, su rol dentro del organismo, la dirección lógica del servicio que se desea consumir y el tipo de política de emisión de token. Por último, se especifica la dirección del STS, la cual, en el ambiente de testing, es siempre https://testservicios.pge.red.uy:6051/TrustServer/SecurityTokenServiceProtected.

```
String userName = "Pruebas";
String role = "ou=gerencia de proyectos,o=agesic";
String service = "http://testservicios.pge.red.uy/timestamp";
String policyName = "urn:tokensimple";
String issuer = "AGESIC";

//Crear un bean con la información para generar el token SAML
RSTBean bean = new RSTBean();
bean.setUserName(userName);
bean.setRole(role);
bean.setService(service);
bean.setPolicyName(policyName);
bean.setIssuer(issuer);

//Definir la url del STS para obtener el token SAML
String stsUrl =
"https://testservicios.pge.red.uy:6051/TrustServer/SecurityTokenServic
eProtected";
```

Figura 8 – Clase PGEClientTest

5. Crear tres *StoreBeans*, como muestra la figura 9, para almacenar los datos de acceso a los almacenes de claves que contienen los certificados y claves requeridas (dos keystores, y un trustore). Las rutas que se especifican en las variables *keyStoreFilePath* y *trustStoreFilePath* apuntan a los archivos *agesictesting_v5.keystore* y *agesictesting_v3.truststore* respectivamente que se encuentran ubicados en la carpeta keystores recientemente creada (en el ambiente de testing, los dos keystores utilizados pueden ser el mismo, mientras que en el ambiente de producción necesariamente deben ser diferentes).

Nota: para consumir un servicio en el ambiente de producción será necesario contar con dos almacenes de certificados digitales: uno para establecer la comunicación mediante SSL, que debe contener un certificado proporcionado por AGESIC específicamente para el Organismo Cliente, y otro para identificar al Organismo Cliente para los cual debe contener el certificado emitido por El Correo para el propio Organismo Cliente. El primero contiene el certificado para establecer una comunicación segura vía SSL con la Plataforma, mientras que el segundo contiene un certificado de Persona Jurídica necesario para firmar las transacciones sobre la Plataforma. En el ambiente de testing, se permite utilizar el mismo certificado digital en ambos casos, el cual es proporcionado por AGESIC y al igual que el truststore se encuentra en el archivo descargado del FTP público de AGESIC; en ambos casos, la contraseña es agesic.

```
//Alias que identifica al certificado que se debe enviar a la PGE
dentro del keystore
String alias = "0f026f823ca3597ced3953188b1628de_be45dff3-4f56-4728-
8332-77080b0c1c08";

String keyStoreSSLFilePath="keystores\\agesictesting_v5.keystore";
String keyStoreSSLPwd = "agesic"; //password del keystore

String keyStoreOrgFilePath="keystores\\agesictesting_v5.keystore";
String keyStoreOrgPwd="agesic"; //password del keystore

String trustStoreFilePath="keystores\\agesictesting_v3.truststore";
String trustStorePwd="agesic"; //password del truststore

StoreBean keyStoreSSL = new StoreBean();
keyStoreSSL.setAlias(alias);
keyStoreSSL.setStoreFilePath(keyStoreSSLFilePath);
keyStoreSSL.setStorePwd(keyStoreSSLPwd);

//En el ambiente de testing se podría usar el mismo bean anterior, en
producción es necesario crear otro, apuntando al keystore del
organismo
StoreBean keyStoreOrg = new StoreBean();
keyStoreOrg.setAlias(alias);
keyStoreOrg.setStoreFilePath(keyStoreOrgFilePath);
keyStoreOrg.setStorePwd(keyStoreOrgPwd);

//El truststore no requiere alias
StoreBean trustStore = new StoreBean();
trustStore.setStoreFilePath(trustStoreFilePath);
trustStore.setStorePwd(trustStorePwd);
```

Figura 9 – Keystore y Truststore

6. Por último, crear una instancia de la clase PGEClient e invocar el método

requestSecurityToken para obtener el *token* SAML firmado por la PGE, como se muestra en la figura 10.

```
PGEClient client = new PGEClient();
SAMLAssertion assertionResponse = null;
try {
  //Solicitar el token SAML pasando los datos de
  //autenticación, los tres beans con los keystores,
  //y la URL del STS
  assertionResponse = client.requestSecurityToken(bean,
            keyStoreSSL, keyStoreOrg, trustStore, stsUrl);
  String stringRepresentation= assertionResponse.toString();
  System.out.println(stringRepresentation);
} catch (Exception e) {
  e.printStackTrace();
  System.exit(1);
}
```

Figura 10 – Obtención del token SAML firmado por la PGE

7. Ejecutar el programa desarrollado hasta ahora, para verificar que el token puede ser obtenido correctamente. Para hacerlo, seleccionar la clase PGEClientTest, hacer clic derecho y luego seleccionar *Run as → Java application*.

8. En caso de ejecutarse correctamente, se desplegará en consola un token SAML.

Nota importante: asegúrese que la hora de su PC se encuentra sincronizada con la hora nacional (de igual manera que los servidores de AGESIC se encuentran sincronizados con ella). Si la hora no coincide, ocurrirá un error en la ejecución ya que la PGE considerará que los mensajes intercambiados no son válidos (estarán vencidos o fechados en el futuro, siendo ambas situaciones invalidantes para obtener el token). En todo caso, si obtiene un error al obtener el token, pruebe ajustando la hora unos minutos antes o después de la hora nacional.

3.5 Invocación al Servicio

Una vez obtenido un *token* SAML firmado por la PGE, es posible consumir el servicio. Para ello, se envía un mensaje SOAP al Servicio Proxy del servicio TimeStamp, el cual debe incluir lo siguiente:

- Servicio y método a invocar (especificados a través de WS-Addressing)
- *Token* SAML firmado por la PGE (incluido a través de WS-Security)
- Información de negocio según el WSDL del servicio (datos a enviar como parámetros de la invocación).

Además, se deben configurar las propiedades para establecer una comunicación segura mediante SSL. En este ejemplo, la invocación al servicio consta de cuatro pasos:

1. Crear las clases para consumir el servicio a partir del WSDL que lo describe. A través de estas clases se creará el mensaje SOAP con la información de negocio.
2. Especificar en el mensaje SOAP el servicio y método a invocar.
3. Adjuntar al mensaje SOAP el *token* SAML firmado por la PGE obtenido en el paso anterior.
4. Configurar las propiedades necesarias para establecer una comunicación SSL.
5. Consumir el servicio.

3.5.1 Crear las clases para consumir el servicio

Para esta tarea se utiliza la herramienta de generación de clientes de Web Services provista por el entorno de desarrollo Eclipse. Los pasos a seguir son los siguientes:

1. Hacer clic derecho en el archivo TimestampServicewsdl.wsdl ubicado en la carpeta wsdl y seleccionar *Web Service → Generate Client* como se muestra en la figura 11.

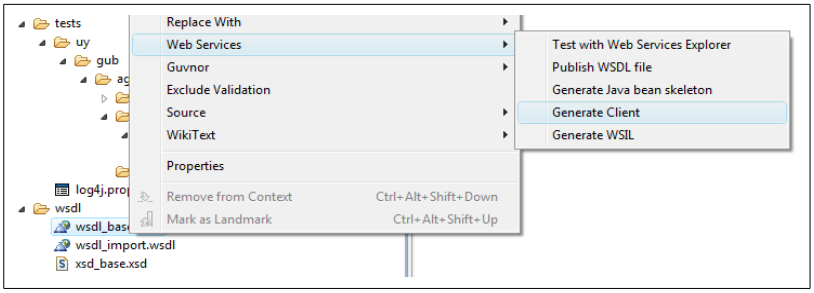


Figura 11 – Generar Clases para Consumir Web Service

2. Seleccionar *JBossWS* como *Web Service Runtime* y seleccionar el nivel de generación del cliente como "*Develop Client*", según se muestra en la figura 12.

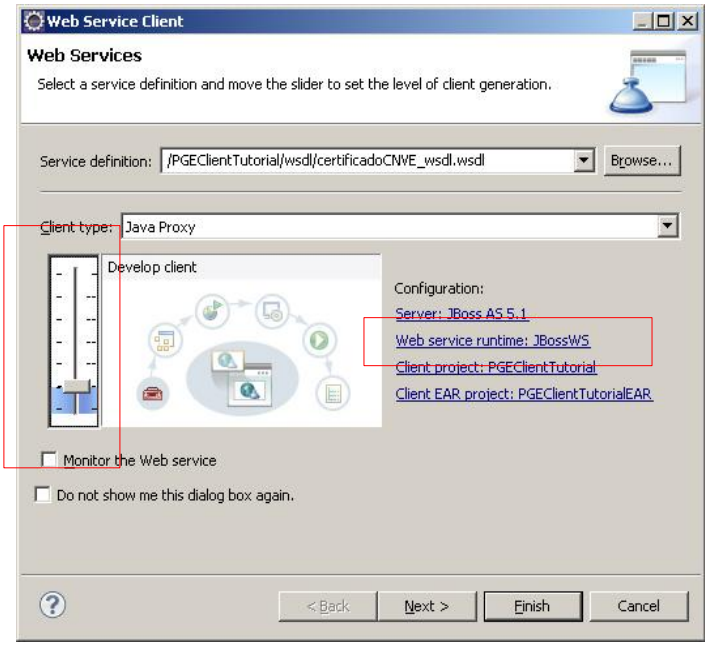


Figura 12 – Generar Clases para Consumir Web Service (parte 2)

3. Presionar el botón "Next" y si se desea, modificar el nombre del paquete donde se colocarán las clases generadas. La figura 13 ilustra el campo donde colocar el nombre del package.

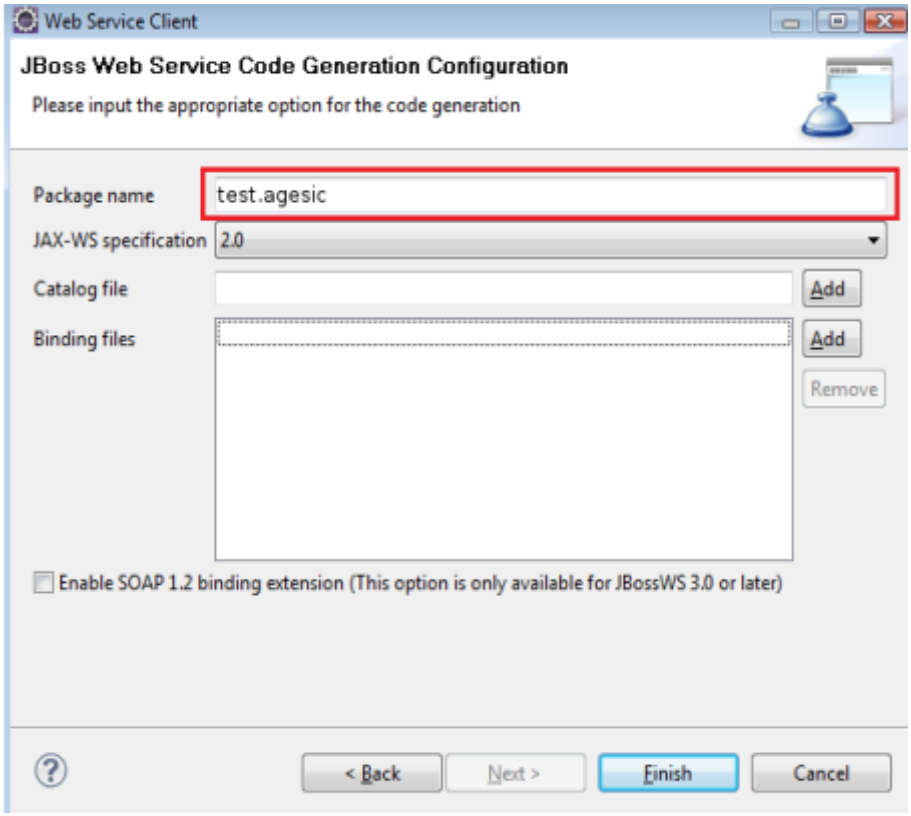


Figura 13 – Generar Clases para Consumir Web Service (parte 3)

Una vez generadas las clases se puede proceder a consumir el servicio. Una vez creada la instancia del servicio a invocar (*timestampService*), se debe obtener el puerto (port), sobre el cual se podrán invocar las operaciones del servicio; cada operación estará representada por un método en el puerto. Sobre éste, se aplicarán las propiedades de WS-Addressing, WS-Security y SSL tal como se ilustrará más adelante.

En la figura 14 se ilustran las líneas de código necesarias para construir el puerto para invocar a el servicio.

```
TimestampService_Service timestampService = new
TimestampService_Service();
                  TimestampService port = timestampService
                              .getTimestampServiceImplPort();
```

Figura 14 – Creación de puerto para invocar al servicio

La figura 15 ilustra qué clases importar.

```
import test.agesic.TimestampService;
import test.agesic.TimestampService_Service;
```

Figura 15 – Creación de puerto para invocar al servicio

3.5.2 Especificar en el mensaje SOAP el servicio y método a invocar.

Como se mencionó anteriormente, la PGE requiere que en la invocación al servicio se especifique el servicio y método a invocar. Para esto, se utilizan los cabezales de WS-Addressing "To" y "Action", respectivamente. La figura 16 muestra cómo especificar esta información utilizando los cabezales WS-Addressing requeridos por la PGE. El valor aplicable al campo "to" debe ser provista por AGESIC, mientras que el valor aplicable al campo "action" puede obtenerse a partir del WSDL, como fue explicado en la tabla 2.

```
//Propiedades para WS-Addressing
AddressingBuilder addrBuilder =
  SOAPAddressingBuilder.getAddressingBuilder();
SOAPAddressingProperties addrProps =
 (SOAPAddressingProperties)addrBuilder.newAddressingProperties();
String actionStr =

"http://www.agesic.gub.uy/soa/TimestampService/TimestampServiceImplPort/Ge
tTimestamp";

addrProps.setTo(new AttributedURIImpl(service));
addrProps.setAction(new AttributedURIImpl(actionStr));

BindingProvider bindingProvider = (BindingProvider)port;
Map<String, Object> reqContext = bindingProvider.getRequestContext();
reqContext.put(JAXWSAConstants.CLIENT_ADDRESSING_PROPERTIES, addrProps);

//Construir la cadena de handlers, en el ordenn especificado
List<Handler> customHandlerChain = new ArrayList<Handler>();
customHandlerChain.add(new WSAddressingClientHandler());
customHandlerChain.add(new WSSecurityHandlerServer());
```

Figura 16 – Agregar los cabezales WS-Addressing al mensaje

La figura 17 muestra como importar las clases a utilizar.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.xml.datatype.XMLGregorianCalendar;

import javax.xml.ws.BindingProvider;
import javax.xml.ws.addressing.AddressingBuilder;
import javax.xml.ws.addressing.AttributedURI;
import javax.xml.ws.addressing.JAXWSAConstants;
import javax.xml.ws.addressing.soap.SOAPAddressingBuilder;
import javax.xml.ws.addressing.soap.SOAPAddressingProperties;
import javax.xml.ws.handler.Handler;

import org.jboss.ws.core.StubExt;
import org.jboss.ws.extensions.addressing.AttributedURIImpl;
import org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler;
import org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer;
```

Figura 17 – Clases a importar para configurar WS-Addressing

3.5.3 Adjuntar en el mensaje SOAP el token SAML firmado por la PGE

Para adjuntar el token SAML utilizando WS-Security se procede de forma similar que para adjuntar los cabezales WS-Addressing. Sin embargo, en este caso AGESIC provee un handler específico (SAMLHandler) para adjuntar el token SAML al mensaje, dado que la plataforma JBoss no provee ninguno prefabricado. La figura 18 presenta cómo utilizar este mecanismo para adjuntar el token SAML requerido por la PGE.

```
//Esto debe colocarse después de los handlers ya configurados
//(WSAddressingClientHandler y WSSecurityHandlerServer), justo antes
//de bindingProvider.getBinding().setHandlerChain(customHandlerChain);
customHandlerChain.add(new SAMLHandler());

bindingProvider.getBinding().setHandlerChain(customHandlerChain);

//Y esto debe colocarse justo debajo de la línea
//reqContext.put(JAXWSAConstants.CLIENT_ADDRESSING_PROPERTIES,addrProps);
reqContext.put(AgesicConstants.SAML1_PROPERTY, assertionResponse);
```

Figura 18 – Agregar token SAML al mensaje usando WS-Security

También se deben importar las clases a usar como se presenta en la figura 19.

```
import uy.gub.agesic.AgesicConstants;
import uy.gub.agesic.jbossws.SAMLHandler;
```

Figura 19 – Importar las clases necesarias para WS-Security

3.5.4 Adjuntar las propiedades necesarias para establecer la comunicación SSL

Para que la invocación al servicio pueda efectuarse a través de SSL deberán configurarse ciertas propiedades en el contexto de la invocación. Estas propiedades harán referencia a los almacenes de claves que se utilizaron para configurar la invocación al STS. En la figura 20 se ilustran las sentencias de código necesarias. Cabe mencionar que para el caso del ejemplo, se utiliza el mismo archivo de keystore que se usó anteriormente (de Organismo) para efectuar la comunicación SSL (esto es válido solo en el ambiente de testing, no así en el ambiente de producción ya que deben utilizarse certificados digitales diferentes, uno emitido por AGESIC y otro por El Correo).

```
//Esto se debe colocar justo después de las dos líneas siguientes:
//reqContext.put(AgesicConstants.SAML1_PROPERTY, assertionResponse);
//reqContext.put(JAXWSAConstants.CLIENT_ADDRESSING_PROPERTIES,addrProps);
reqContext.put(StubExt.PROPERTY_AUTH_TYPE,StubExt.PROPERTY_AUTH_TYPE_WSSE);
reqContext.put(StubExt.PROPERTY_KEY_STORE, keyStoreSSLFilePath);
reqContext.put(StubExt.PROPERTY_KEY_STORE_PASSWORD, keyStoreSSLPwd);
reqContext.put(StubExt.PROPERTY_TRUST_STORE, trustStoreFilePath);
reqContext.put(StubExt.PROPERTY_TRUST_STORE_PASSWORD, trustStorePwd);


//Nota: lo anterior puede ser sustituido también por el siguiente código:
System.setProperty("javax.net.ssl.keyStore",
  keyStoreSSLFilePath);
System.setProperty("javax.net.ssl.keyStorePassword",
  sslKeyStore.getStorePwd());
System.setProperty("javax.net.ssl.trustStore",
  sslTrustStore.getStoreFilePath());
System.setProperty("javax.net.ssl.trustStorePassword",
  sslTrustStore.getStorePwd());
```

Figura 20 – Configuración de propiedades para establecer la comunicación SSL

3.5.5 Consumir el Servicio

Por último, se debe invocar el servicio. Para ello se debe agregar el código de la siguiente figura e importar las clases a utilizar como se presenta a continuación.

```
//Crear los parámetros de entrada
//Estas clases fueron generadas en el paso 1, cuando se importó el WSDL
XMLGregorianCalendar timestamp = port.getTimestamp();
System.out.println(timestamp.toString());
```

3.5.6 Probar el cliente programado

Para ejecutar el cliente implementado, seleccionar la clase PGEClientTest, hacer clic derecho y ejecutar *Run as → Java Application*.

En el *Apéndice 1* se puede ver el código fuente completo.

4 Invocación de un servicio que requiere autenticación con WS-Security

Algunos servicios disponibles a través de la PGE también pueden ser accedidos por fuera de la PGE, directamente a través de la RedUy. En estos casos, es común que el proveedor del servicio requiera que el cliente especifique un nombre de usuario y una contraseña, mediante WS-Security. Luego, cuando se expone el servicio a través de la PGE, queda una doble autenticación: ante la PGE y ante el proveedor del servicio, lo que exige que el mensaje SOAP envíe dos cabeceras de WS-Security. Para que la PGE, que es la que hace de intermediaria, pueda identificar cuál de las dos es la que debe procesar, se define que la cabecera orientada a la PGE deba estar destinada al actor "**http://testservicios.pge.red.uy/wsproxy**" (la otra cabecera puede no especificar actor, o especificar cualquier otro actor excepto "http://testservicios.pge.red.uy/wsproxy"). A continuación se muestran los cambios que deben efectuarse sobre el código antes descrito para incluir la nueva cabecera, incluyendo el nombre de usuario y la contraseña, los cuales deben ser proporcionados por el proveedor del servicio directamente al consumidor (AGESIC no gestiona esta segunda autenticación, la cual corre por cuenta exclusiva del proveedor del servicio y la tramitación para obtener un par de valores válidos debe ser realizada por el cliente directamente ante el proveedor del servici).

Nota: para poder utilizar lo que se explica a continuación, es imprescindible contar con la versión 1.5 o posterior del adaptador PGEClient.jar.

4.1 Verificación de que se requiere usuario y contraseña

Para determinar que efectivamente se requiere enviar un usuario y contraseña en el mensaje SOAP, se debe analizar el WSDL que se utiliza para crear los clientes, y determinar si existe una definición de política, la cual se reconoce por el tag wsp:Policy, y una referencia a ella en el service, como se muestra en la figura 26:

```
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Aclaraciones">
    <wsdl:port binding="tns:AclaracionesSoapBinding" name="AclaracionesPort">
      <soap:address location="https://testservicios.pge.red.uy:6356/sicews/aclaraciones" />
      <wsp:PolicyReference URI="#UsernameTokenPolicy" />
    </wsdl:port>
  </wsdl:service>
  <wsp:Policy wsu:Id="UsernameTokenPolicy" 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">
    <wsp:ExactlyOne>
      <wsp:All>
        <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
          <wsp:Policy>
            <sp:UsernameToken
              sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
              <wsp:Policy>
                <sp:WssUsernameToken10 />
              </wsp:Policy>
            </sp:UsernameToken>
          </wsp:Policy>
        </sp:SupportingTokens>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
</wsdl:definitions>
```

Figura 21 – Determinar si existe una política del proveedor

4.2 Inclusión de usuario y contraseña en el mensaje SOAP

Si el proveedor del servicio exige que el cliente envíe un usuario y una contraseña, como se mostró anteriormente, entonces debe procederse de la siguiente manera:

4.2.1 Añadir los handlers necesarios

En el mismo lugar donde antes se añadían los handlers (WSAddressingClientHandler y SAMLHandler), añadir también y a continuación dos handlers más (org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer y uy.gub.agesic.jbossws.WSSecurityUsernamePasswordHandler, se marca en negrita el nuevo código), como se muestra en la figura 27:

```
List<Handler> customHandlerChain = new ArrayList<Handler>();
customHandlerChain.add(new WSAddressingClientHandler());
customHandlerChain.add(new SAMLHandler());
customHandlerChain.add(new WSSecurityHandlerServer());
customHandlerChain.add(new WSSecurityUsernamePasswordHandler());
bindingProvider.getBinding().setHandlerChain(customHandlerChain);
```

Figura 22 – Inclusión de handlers para WS-Security

La figura 28 muestra los imports necesarios.

```
import org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer;
import uy.gub.agesic.jbossws.WSSecurityUsernamePasswordHandler;
```

Figura 23 – Imports necesarios para los handlers

4.2.2 Especificar usuario, contraseña y actor

Para especificar el usuario y la contraseña para poder invocar el servicio (datos que debieron ser proporcionados por el proveedor del servicio), se deben agregar las siguientes propiedades justo a continuación de las anteriores, como lo muestra la figura 29:

- AgesicConstants.SAML_ACTOR: debe ser el actor reconocido por la PGE. Dado que el mensaje SOAP contendrá dos cabeceras WS-Security, una de ellas debe ser marcada con destino a la PGE (la otra pasará hasta el proveedor del servicio). Actualmente, el actor reconocido por la PGE es "http://testservicios.pge.red.uy/wsproxy".
- BindingProvider.USERNAME_PROPERTY y BindingProvider.PASSWORD_PROPERTY: deben ser el nombre de usuario y la contraseña requeridos por el proveedor del servicio para permitir la invocación de alguna operación del mismo. Estos valores deben ser proporcionados directamente por el proveedor del servicio.

```
reqContext.put(AgesicConstants.SAML_ACTOR,
                            "http://testservicios.pge.red.uy/wsproxy");
reqContext.put(BindingProvider.USERNAME_PROPERTY, "usuario");
reqContext.put(BindingProvider.PASSWORD_PROPERTY, "password");
```

Figura 24 – Imports necesarios para los handlers

5 Apéndices

5.1 Apéndice 1 – Endorsado de bibliotecas

En algunos casos, es posible que se requiera endorsar algunas bibliotecas para que ciertas clases tomen precedencia sobre otras; en particular, el adaptador PGEClient.jar requiere que las bibliotecas que se encuentran en la carpeta endorsed del archivo que fue descargado del FTP público de AGESIC tomen precedencia respecto de las que pudieran existir en el entorno donde se ejecuta el cliente. Es necesario recurrir al mecanismo de endorsado cuando al intentar consumir un servicio web a través de la PGE, se obtiene un error con el siguiente mensaje:

```
OpenSAML requires an xml parser that supports JAXP 1.3 and DOM3.

The JVM is currently configured to use the Sun XML parser, which is known to be
buggy and can not be used with OpenSAML.  Please endorse a functional JAXP
library(ies) such as Xerces and Xalan.  For instructions on how to endorse a new
parser see http://java.sun.com/j2se/1.5.0/docs/guide/standards/index.html
```

Para utilizar el mecanismo de endorsado, existen dos alternativas:

- Copiar todos las librerías que se desean endorsar en el directorio lib/endorsed de la instalación del entorno de ejecución de Java (JRE). Por ejemplo, si el JRE está instalado en C:\Java\JRE, entonces, las librerías deben copiarse a C:\Java\JRE\lib\endorsed. Si el directorio endorsed no existe, se debe crearlo. Este mecanismo afecta a todas las aplicaciones que utilicen el JRE, por lo que puede tener efectos secundarios sobre otras aplicaciones. No es un mecanismo recomendado.
- Especificar, al momento de ejecutar el cliente, el directorio de endorsado, es decir, el directorio que contiene a las librerías que se desean endorsar. Esto afecta solo a la aplicación particular. Para especificar el directorio de endorsado, se debe ejecutar el cliente especificando el parámetro -Djava.endorsed.dirs=<ruta_a_la_carpeta>; alternativamente, también se puede hacer en el código fuente mismo de la aplicación, ANTES de realizar cualquier actividad relacionada con el servicio web que se desea consumir, con la sintaxis System.setProperty("java.endorsed.dirs",<ruta_a_la_carpeta>);

Nota: en el caso de que el cliente no sea standalone, sino que funcione dentro de una aplicación que se ejecuta en un servidor JBoss, el mecanismo de endorsado consiste simplemente de copiar todos los archivos que se desean endorsar a la carpeta <jboss>/lib/endorsed. Si la carpeta endorsed no existe, se debe crearla. Recordar que cualquier cambio que se realice sobre las librerías en JBoss requieren que se reinicie el servidor de aplicaciones; el mecanismo de endorsado no es una excepción. También tener en cuenta que este mecanismo afectará a todas las aplicaciones que estén deployadas en el servidor JBoss.

5.2 Apéndice 2 – Consumo sin WS-Security

```
package test;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.xml.datatype.XMLGregorianCalendar;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.addressing.AddressingBuilder;
```

```
import javax.xml.ws.addressing.AttributedURI;
import javax.xml.ws.addressing.JAXWSAConstants;
import javax.xml.ws.addressing.soap.SOAPAddressingBuilder;
import javax.xml.ws.addressing.soap.SOAPAddressingProperties;
import javax.xml.ws.handler.Handler;

import org.jboss.ws.core.StubExt;
import org.jboss.ws.extensions.addressing.AttributedURIImpl;
import org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler;

import test.agesic.TimestampService;
import test.agesic.TimestampService_Service;
import uy.gub.agesic.AgesicConstants;
import uy.gub.agesic.beans.RSTBean;
import uy.gub.agesic.beans.SAMLAssertion;
import uy.gub.agesic.beans.StoreBean;
import uy.gub.agesic.jbossws.SAMLHandler;
import uy.gub.agesic.sts.client.PGEClient;

public class Tutorial {

      public static void main(String[] args) {

            String userName = "Pruebas";
            String role = "ou=gerencia de proyectos,o=agesic";
            String service = "http://testservicios.pge.red.uy/timestamp";
            String policyName = "urn:tokensimple";
            String issuer = "AGESIC";
            RSTBean bean = new RSTBean();
            bean.setUserName(userName);
            bean.setRole(role);
            bean.setService(service);
            bean.setPolicyName(policyName);

            bean.setIssuer(issuer);

            String stsUrl =
"https://testservicios.pge.red.uy:6051/TrustServer/SecurityTokenServiceProtecte
d";

            String alias = "0f026f823ca3597ced3953188b1628de_be45dff3-4f56-
4728-8332-77080b0c1c08";
            String keyStoreSSLFilePath = "keystores/agesictesting_v5.keystore";
            String keyStoreSSLPwd = "agesic";

            String keyStoreOrgFilePath = "keystores/agesictesting_v5.keystore";
            String keyStoreOrgPwd = "agesic";
            String trustStoreFilePath =
"keystores/agesictesting_v3.truststore";
            String trustStorePwd = "agesic";

            StoreBean keyStoreSSL = new StoreBean();
            keyStoreSSL.setAlias(alias);
            keyStoreSSL.setStoreFilePath(keyStoreSSLFilePath);
            keyStoreSSL.setStorePwd(keyStoreSSLPwd);

            StoreBean keyStoreOrg = new StoreBean();
            keyStoreOrg.setAlias(alias);
            keyStoreOrg.setStoreFilePath(keyStoreOrgFilePath);
```

```
            keyStoreOrg.setStorePwd(keyStoreOrgPwd);

            StoreBean trustStore = new StoreBean();
            trustStore.setStoreFilePath(trustStoreFilePath);
            trustStore.setStorePwd(trustStorePwd);

            PGEClient client = new PGEClient();
            SAMLAssertion assertionResponse = null;
            try {
                  assertionResponse = client.requestSecurityToken(bean,
keyStoreSSL,
                              keyStoreOrg, trustStore, stsUrl);

                  System.out.println(assertionResponse.toString());

                  TimestampService_Service timestampService = new
TimestampService_Service();
                  TimestampService port = timestampService
                              .getTimestampServiceImplPort();

                  // Configurar WS-Addressing
                  AddressingBuilder addrBuilder = SOAPAddressingBuilder
                              .getAddressingBuilder();
                  SOAPAddressingProperties addrProps =
(SOAPAddressingProperties) addrBuilder
                              .newAddressingProperties();
                  String actionStr =
"http://www.agesic.gub.uy/soa/TimestampService/TimestampServiceImplPort/GetTime
stamp";

                  AttributedURI to = new AttributedURIImpl(service);
                  AttributedURI action = new AttributedURIImpl(actionStr);
                  addrProps.setTo(to);
                  addrProps.setAction(action);

                  BindingProvider bindingProvider = (BindingProvider) port;
                  Map<String, Object> reqContext = bindingProvider
                              .getRequestContext();
                  reqContext.put(AgesicConstants.SAML1_PROPERTY,
assertionResponse);

                  reqContext.put(JAXWSAConstants.CLIENT_ADDRESSING_PROPERTIES,
                              addrProps);
                  reqContext.put(StubExt.PROPERTY_AUTH_TYPE,
                              StubExt.PROPERTY_AUTH_TYPE_WSSE);
                  reqContext.put(StubExt.PROPERTY_KEY_STORE,
keyStoreSSLFilePath);
                  reqContext.put(StubExt.PROPERTY_KEY_STORE_PASSWORD,
keyStoreSSLPwd);
                  reqContext.put(StubExt.PROPERTY_TRUST_STORE,
trustStoreFilePath);
                  reqContext
                              .put(StubExt.PROPERTY_TRUST_STORE_PASSWORD,
trustStorePwd);

                  // Cadena de handlers
                  List<Handler> customHandlerChain = new ArrayList<Handler>();
                  customHandlerChain.add(new WSAddressingClientHandler());
                  customHandlerChain.add(new SAMLHandler());
```

```
      bindingProvider.getBinding().setHandlerChain(customHandlerChain);

                  // Invocar al servicio
                  XMLGregorianCalendar timestamp = port.getTimestamp();
                  System.out.println(timestamp.toString());

            } catch (Exception e) {
                  e.printStackTrace();
                  System.exit(1);
            }
      }

}
```

5.3 Apéndice 3 – Consumo con WS-Security

Nota: el siguiente código es solo a modo de ejemplo, ya que la operación y el servicio utilizados no requieren de usuario y contraseña para su invocación.

```
package test;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.xml.datatype.XMLGregorianCalendar;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.addressing.AddressingBuilder;
import javax.xml.ws.addressing.AttributedURI;
import javax.xml.ws.addressing.JAXWSAConstants;
import javax.xml.ws.addressing.soap.SOAPAddressingBuilder;
import javax.xml.ws.addressing.soap.SOAPAddressingProperties;
import javax.xml.ws.handler.Handler;

import org.jboss.ws.core.StubExt;
import org.jboss.ws.extensions.addressing.AttributedURIImpl;
import org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler;
import org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer;

import test.agesic.TimestampService;
import test.agesic.TimestampService_Service;
import uy.gub.agesic.AgesicConstants;
import uy.gub.agesic.beans.RSTBean;
import uy.gub.agesic.beans.SAMLAssertion;
import uy.gub.agesic.beans.StoreBean;
import uy.gub.agesic.jbossws.SAMLHandler;
import uy.gub.agesic.jbossws.WSSecurityUsernamePasswordHandler;
import uy.gub.agesic.sts.client.PGEClient;

public class Tutorial {

      public static void main(String[] args) {

            String userName = "Pruebas";
            String role = "ou=gerencia de proyectos,o=agesic";
            String service = "http://testservicios.pge.red.uy/timestamp";
            String policyName = "urn:tokensimple";
```

```
            String issuer = "AGESIC";
            RSTBean bean = new RSTBean();
            bean.setUserName(userName);
            bean.setRole(role);
            bean.setService(service);
            bean.setPolicyName(policyName);

            bean.setIssuer(issuer);

            String stsUrl =
"https://testservicios.pge.red.uy:6051/TrustServer/SecurityTokenServiceProtecte
d";

            String alias = "0f026f823ca3597ced3953188b1628de_be45dff3-4f56-
4728-8332-77080b0c1c08";
            String keyStoreSSLFilePath = "keystores/agesictesting_v5.keystore";
            String keyStoreSSLPwd = "agesic";

            String keyStoreOrgFilePath = "keystores/agesictesting_v5.keystore";
            String keyStoreOrgPwd = "agesic";
            String trustStoreFilePath =
"keystores/agesictesting_v3.truststore";
            String trustStorePwd = "agesic";

            StoreBean keyStoreSSL = new StoreBean();
            keyStoreSSL.setAlias(alias);
            keyStoreSSL.setStoreFilePath(keyStoreSSLFilePath);
            keyStoreSSL.setStorePwd(keyStoreSSLPwd);

            StoreBean keyStoreOrg = new StoreBean();
            keyStoreOrg.setAlias(alias);
            keyStoreOrg.setStoreFilePath(keyStoreOrgFilePath);
            keyStoreOrg.setStorePwd(keyStoreOrgPwd);

            StoreBean trustStore = new StoreBean();
            trustStore.setStoreFilePath(trustStoreFilePath);
            trustStore.setStorePwd(trustStorePwd);

            PGEClient client = new PGEClient();
            SAMLAssertion assertionResponse = null;
            try {
                  assertionResponse = client.requestSecurityToken(bean,
keyStoreSSL,
                              keyStoreOrg, trustStore, stsUrl);

                  System.out.println(assertionResponse.toString());

                  TimestampService_Service timestampService = new
TimestampService_Service();
                  TimestampService port = timestampService
                              .getTimestampServiceImplPort();

                  // Configurar WS-Addressing
                  AddressingBuilder addrBuilder = SOAPAddressingBuilder
                              .getAddressingBuilder();
                  SOAPAddressingProperties addrProps =
(SOAPAddressingProperties) addrBuilder
                              .newAddressingProperties();
                  String actionStr =
```

```
"http://www.agesic.gub.uy/soa/TimestampService/TimestampServiceImplPort/GetTime
stamp";

                  AttributedURI to = new AttributedURIImpl(service);
                  AttributedURI action = new AttributedURIImpl(actionStr);
                  addrProps.setTo(to);
                  addrProps.setAction(action);

                  BindingProvider bindingProvider = (BindingProvider) port;
                  Map<String, Object> reqContext = bindingProvider
                              .getRequestContext();
                  reqContext.put(AgesicConstants.SAML1_PROPERTY,
assertionResponse);

                  reqContext.put(JAXWSAConstants.CLIENT_ADDRESSING_PROPERTIES,
                              addrProps);
                  reqContext.put(StubExt.PROPERTY_AUTH_TYPE,
                              StubExt.PROPERTY_AUTH_TYPE_WSSE);
                  reqContext.put(StubExt.PROPERTY_KEY_STORE,
keyStoreSSLFilePath);
                  reqContext.put(StubExt.PROPERTY_KEY_STORE_PASSWORD,
keyStoreSSLPwd);
                  reqContext.put(StubExt.PROPERTY_TRUST_STORE,
trustStoreFilePath);
                  reqContext
                              .put(StubExt.PROPERTY_TRUST_STORE_PASSWORD,
trustStorePwd);

                  // Cadena de handlers
                  List<Handler> customHandlerChain = new ArrayList<Handler>();
                  customHandlerChain.add(new WSAddressingClientHandler());
                  customHandlerChain.add(new SAMLHandler());

      bindingProvider.getBinding().setHandlerChain(customHandlerChain);

            // WS-Security
                  customHandlerChain.add(new WSSecurityHandlerServer());
                  customHandlerChain.add(new
WSSecurityUsernamePasswordHandler());
                  reqContext.put(BindingProvider.USERNAME_PROPERTY, "usuario");
                  reqContext.put(BindingProvider.PASSWORD_PROPERTY,
"password");

                  // Invocar al servicio
                  XMLGregorianCalendar timestamp = port.getTimestamp();
                  System.out.println(timestamp.toString());

            } catch (Exception e) {
                  e.printStackTrace();
                  System.exit(1);
            }
      }

}
```

5.4 Apéndice 4 – Determinar el valor del atributo soap:Action para un servicio

Como se explicó a lo largo del tutorial, para invocar el servicio es necesario especificar la URL lógica del mismo, y el identificador de la operación, los cuales deben ser enviados en sendos cabezales SOAP denominados "to" y "action" respectivamente. El cabezal "to" es asignado por AGESIC, y por tanto será comunicado al cliente. El cabezal "action" debe ser obtenido a partir del documento WSDL que describe el servicio, como se explica a continuación:

1. Abrir el archivo con extensión wsdl que describe el servicio.
2. Buscar el tag "binding" dentro del archivo; dentro de este tag se listan todas las operaciones que ofrece el servicio.
3. Buscar dentro del tag "binding" el tag "operation" cuyo atributo "name" corresponda con la operación que se desea invocar.
4. Observar el valor del atributo "soapAction" del tag "soap:binding" que se encuentra inmediatamente a continuación del tag "operation" identificado en el paso anterior.
5. Determinar, según el valor del atributo "soapAction" el texto que se debe enviar como valor del cabezal "action", de la siguiente manera:
   - Si el valor del atributo "soapAction" es vacío, se debe especificar el valor del atributo "to" concatenado con el nombre de la operación, es decir, el mismo valor del atributo "name" del tag "operation" (separados mediante una barra común).
   - Si el valor del atributo "soapAction" no es vacío, se debe especificar exactamente dicho valor, respetando mayúsculas y minúsculas.

Ejemplos:

Ejemplo 1:

Suponga que el valor del atributo soap:to es:

"http://testservicios.pge.red.uy/msp/certificadoCNVE" y el WSDL especifica lo siguiente para la operación "registrarCNVE":

```
<wsdl:operation name="registrarCNVE">
   <soap:operation soapAction="" style="document" />
   ...
</wsdl:operation>
```

Entonces, en este caso, se debe especificar:

"http://testservicios.pge.red.uy/msp/certificadoCNVE/registrarCNVE".

Ejemplo 2:

Sunpoga que el WSDL especifica lo siguiente para la operación "registrarCNVE":

```
<wsdl:operation name="registrarCNVE">
   <soap:operation
soapAction="http://xml.cnve.msp.gub.uy/wsdl/certificadoCNVEWSDL/
certificadoCNVEWSDLPortType/registrarCNVE" style="document" />
   ...
</wsdl:operation>
```

En este caso se debe especificar "http://xml.cnve.msp.gub.uy/wsdl/certificadoCNVEWSDL/ certificadoCNVEWSDLPortType/registrarCNVE".

La siguiente imagen ilustra el tag soapAction que se debe examinar en el archivo WSDL para otener el valor para el campo “to”:

```
<wsdl:operation name="registrarCNVE">
    <soap:operation
        soapAction="http://xml.cnve.msp.gub.uy/wsdl/certificadoCNVEWSDL/certificadoCNVEWSDLPortType/registrarCNVE"
        style="document" />
    <wsdl:input name="input1">
        <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output name="output1">
        <soap:body use="literal" />
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getCertificadosByCriteria">
```

6 Referencias

[1] - http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[2] - http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.doc

[3] - http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf