Capítulo IV

Guía de Programación Java para la Plataforma de Gobierno Electrónico

Introducción

Este capítulo brinda guías de desarrollo para la implementación de servicios y aplicaciones cliente de la PGE, utilizando Java.

En primera instancia se describe cómo utilizar una Librearía de Ejemplo, desarrollada por AGESIC, que muestra cómo obtener un token de seguridad firmado digitalmente por la PGE.

Luego se provee una guía, paso a paso, para el desarrollo de una aplicación Java de escritorio (Aplicación Cliente) que consuma un servicio de la PGE. Para esto se utiliza Eclipse [1] y JBoss Tools [2] como entorno de desarrollo, y JBossWS - Native [3] como stack de Web Services. Asimismo, se utiliza la Librearía de Ejemplo desarrollada por AGESIC.

Se recomienda leer previamente la Descripción Técnica de la PGE en el Capítulo III.

Librería de Ejemplo AGESIC

AGESIC desarrolló una librería Java para brindar un ejemplo de cómo obtener un *token* de seguridad firmado digitalmente por la PGE. En esta sección se describen las principales características de la librería y cómo utilizarla.

Importante: La librería fue desarrollada como prueba de concepto, por lo cual no está garantizada la ausencia de errores, ni fallas de seguridad. No se recomienda entonces utilizarla en producción, sin los resguardos apropiados según las políticas de *testing* y seguridad de cada organismo.

Descripción General

La Librería de Ejemplo resuelve las siguientes dos tareas:

- la solicitud y obtención de un *token* de seguridad SAML firmado digitalmente por el organismo
- la solicitud y obtención de un *token* de seguridad SAML firmado digitalmente por la PGE

Para realizar estas tareas, la librería provee la clase PGEClient, en el paquete uy.gub.agesic.sts.client. Dicha clase cuenta con el método

requestSecurityToken que es el encargado de invocar al STS de la PGE y retornar el *token* de seguridad emitido por éste.

Internamente, el método realiza las siguientes tareas:

- emite un *token* SAML firmado por el Organismo Cliente
- genera y envía al STS de la PGE un mensaje RequestSecurityToken (RST), siguiendo el estándar WS-Trust, en el que se incluye el *token* emitido y otros datos requeridos
- al recibir el mensaje de respuesta (RequestSecurityTokenResponse, RSTR), se comprueba que el *token* SAML recibido fue emitido por la PGE mediante la verificación de su firma digital

Obtención del token SAML firmado por la PGE

Para obtener un token SAML firmado por la PGE se debe crear una instancia de la clase PGEClient e invocar el método requestSecurityToken, el cual recibe tres parámetros: bean, keyStore y trustStore.

El parámetro bean (uy.gub.agesic.beans.RSTBean) aloja los datos que se utilizan para emitir el token firmado por el Organismo Cliente y para construir el mensaje RST a enviar al STS. Concretamente estos datos son: nombre de usuario, rol del usuario, servicio, issuer y policy name.

El parámetro keystore (uy.gub.agesic.beans.StoreBean) aloja los datos para acceder a la keystore donde se almacenan las claves y certificados digitales del organismo. Estos se utilizan para firmar el token de seguridad emitido y para establecer la conexión SSL. La Tabla 1 describe los datos que se deben especificar en este parámetro y los métodos de la clase StoreBean a utilizar para este fin.

Nombre	Método	Descripción
Alias	setAlias	Alias de la entidad en la *keystore*.
Ruta	setStoreFilePath	Ruta del archivo de la *keystore*
Contraseña	setStorePwd	Contraseña para acceder a la *keystore*.

Tabla 1 – Datos a Especificar en el parámetro keyStore

De forma similar, el parámetro trustStore (uy.gub.agesic.beans.StoreBean) aloja los datos para acceder a la trustStore donde se almacenan los certificados de la PGE. Estos certificados se utilizan para verificar la firma del token emitido por el STS y para establecer la conexión SSL. La Tabla 2 describe los datos que se deben especificar en este parámetro y los métodos de la clase StoreBean a utilizar para este fin.

Nombre	Método	Descripción
Ruta	setStoreFilePath	Ruta del archivo de la *trustStore*
Contraseña	setStorePwd	Contraseña para acceder a la *trustStore*.

Tabla 2 – Datos a Especificar en el parámetro trustStore

La Figura 1 presenta un ejemplo completo donde se invoca el método requestSecurityToken y se obtiene un uy.gub.agesic.beans.SAMLAssertion con el token de seguridad emitido por la PGE.

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

StoreBean keyStore = new StoreBean();
keyStore.setAlias(alias);
keyStore.setStoreFilePath(keyStoreFilePath);
keyStore.setStorePwd(keyStorePwd);

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

PGEClient client = new PGEClient();
SAMLAssertion assertionResponse =
client.requestSecurityToken(bean, keyStore,
trustStore);
```

Figura 1 – Obtener un *token* firmado por la PGE

La documentación Java de la librería se puede acceder en [4].

Tutorial: Consumir un Servicio de la PGE

Uno de los principales escenarios de uso de la PGE, es el consumo de servicios. En esta sección se presenta un tutorial que explica cómo consumir un servicio de la PGE utilizando Java.

Objetivo

El objetivo de este tutorial es proveer una guía, paso a paso, para el desarrollo de una aplicación Java de escritorio (Aplicación Cliente) que consuma un servicio de la PGE. Para esto se utiliza Eclipse y JBoss Tools como entorno de desarrollo, y JBoss WS – Native como *stack* de Web Services.

Prerrequisitos

Para implementar y ejecutar la Aplicación Cliente se debe cumplir con los prerrequisitos que se describen en esta sección.

Conocimientos Requeridos

Para comprender el tutorial se requiere que el lector esté familiarizado con el desarrollo de aplicaciones Java EE, haya desarrollado Web Services con tecnología Java y cuente con conocimientos de seguridad informática. Concretamente, se asume que el lector conoce los estándares WS-Addressing, WS-Security, WS-Trust y SAML, y tiene experiencia en el uso de certificados digitales.

Si no se cuenta con estos conocimientos, se puede consultar el marco técnico que se presenta en el Apéndice 1 y la bibliografía asociada.

Conectividad con la PGE

Para poder ejecutar en un organismo la Aplicación Cliente se requiere que:

- el organismo esté conectado a la REDuy

- los firewalls de REDuy estén configurados para habilitar el tráfico de red requerido
- se puedan establecer conexiones SSL entre el organismo y la PGE

Si no se cumple con alguno de estos prerrequisitos consultar la sección "**¡Error! No se encuentra el origen de la referencia.**" del Capítulo III.

Requerimientos de Software

La Tabla 3 presenta las herramientas y productos de *software* requeridos para desarrollar y ejecutar la Aplicación Cliente.

Producto	Versión
Java Developer Kit (JDK)	5.0, Update 22
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 3 – Requerimientos de Software

Almacenes de Claves y Certificados

La ejecución del escenario requiere una *keyStore* y una *trustStore* que almacene los certificados y claves necesarias para establecer la conexión SSL, firmar los tokens SAML emitidos y verificar la firma de los *tokens* de seguridad emitidos por la PGE.

En la *keystore* se deben alojar las claves y certificados para:

- Firmar los *tokens* SAML emitidos.
- Llevar a cabo la comunicación SSL.

Por otro lado, la *truststore* necesita tener la siguiente información:

- Certificado de la CA de la PGE, utilizada para llevar a cabo la comunicación SSL.
- Certificado de la PGE para verificar la firma de los *tokens* SAML emitidos por la misma.

En [4] se pueden encontrar certificados, *truststores* y *keystores* de ejemplo. Por más información acerca de cómo generar un *keystore* y *truststore* en Java, ver [5]. Por información acerca de cómo importar un archivo pfx a la *keystore*, ver [6].

Descripción del Escenario

La Figura 2 presenta el escenario de ejemplo que se utiliza en este tutorial, en el cual intervienen dos organismos: el Banco de Previsión Social (BPS) (Organismo Cliente) y el Ministerio de Salud Pública (MSP) (Organismo Proveedor).

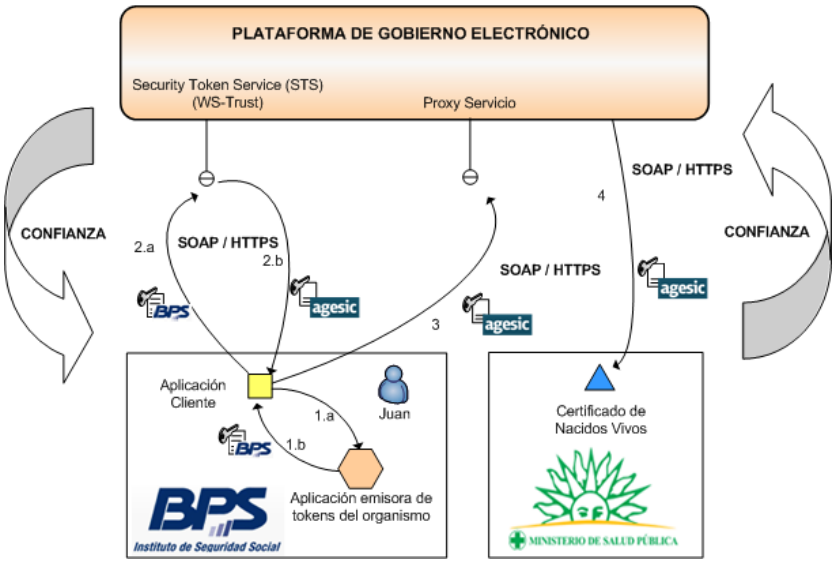

Figura 2 – Escenario de Uso de la PGE

El MSP provee el servicio "Certificado de Nacidos Vivos" el cual tiene dos métodos: "getCertificadosByCriteria" y "registrarCNEV". Cuando se registró el servicio en la PGE, se desplegó un Servicio Proxy en ella para que las Aplicaciones Cliente accedieran al servicio a través de él. Además, mediante la configuración de políticas de control de acceso, el MSP autorizó a los usuarios con rol "doctor" de la sección "prestaciones" del BPS a consumir el método "registrarCNEV" de dicho servicio.

Por otro lado, en el BPS hay una Aplicación Cliente que está siendo utilizada por el usuario Juan que tiene el rol "doctor" en la sección "prestaciones". La aplicación necesita acceder al servicio del MSP para lo cual, utilizando las credenciales del usuario Juan 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).

Luego con dicho *token* obtiene del STS de la PGE, a través del 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=doctor, ou=prestaciones, o=bps".

Por último, la Aplicación Cliente invoca al Servicio del MSP mediante su Servicio Proxy. En la invocación se incluye el *token* firmado por la PGE y se especifican el servicio (Certificado de Nacidos Vivos) y método (registrarCNEV) a invocar. Dado que el usuario Juan está autorizado a utilizar el método del servicio, la invocación se efectúa de forma exitosa.

La Tabla 4 especifica algunos de los datos a utilizar en la implementación del escenario.

Dato	Valor
Nombre de Usuario	Juan
Rol de Usuario	ou=doctor, ou=prestaciones, o=bps
Dirección Lógica del Servicio	http://192.168.40.190:9000/Servicio
Método del Servicio	http://xml.cnve.msp.gub.uy/wsdl/certificadoCNVEWSDL/certificadoCNVEWSDLPortType/registrarCNVE
PolicyName[1]	urn:simpletoken
Tipo de *Token*[2]	http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1

Tabla 4 – 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 dirección del Servicio Proxy a donde el cliente debe enviar los mensajes SOAP para invocar al servicio.

Implementación Escenario

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

[1] Es la política de autenticación utilizada por AGESIC para la verificación de solicitudes del cliente. Actualmente el único valor posible es "urn:simpletoken)".

[2] Actualmente la PGE acepta la emisión de *tokens* SAML versión 1.1.

La implementación del escenario comprende tres etapas:

- Creación del Proyecto Eclipse y Configuración del Entorno
- Obtención del *token* de Seguridad emitido por la PGE
- Invocación al Servicio

En las siguientes sub-secciones se describen en detalle estas etapas.

Creación y Configuración del Proyecto Eclipse

La primera etapa consta de la creación de un proyecto Eclipse y su configuración, así como la del entorno de desarrollo. Concretamente en esta etapa se debe: crear un proyecto Eclipse, crear un Runtime para JBossAS e incluir librerías y otros archivos necesarios en el proyecto creado.

Creación del Proyecto Eclipse

Una vez iniciado Eclipse, crear un proyecto de tipo "Faceted Project"[3] incluyendo los facets Java 5.0, JBoss Web Service Core 3.0 y Dynamic Web Module.

Nota: La aplicación Java a implementar no es una aplicación Web, sin embargo, se incluye el *facet* Dynamic Web Module porque es requerido por el *facet* JBoss Web Service Core 3.0.

Los pasos a seguir para realizar esta tarea son:

1. Seleccionar *File* → *New* → *Other* → *General* → *Faceted Project*, crear un nuevo proyecto con el nombre PGEClientTutorial y los facets Java 5.0, JBoss Web Service Core 3.0 y Dynamic Web Module 2.4 según la Figura 3 y Figura 4.

   Nota: Para cada *facet* seleccionado se deben configurar algunos valores que se describen en los siguientes pasos.

[3] Los *Faceted Projects* de Eclipse son proyectos que aceptan unidades de funcionalidad (*facets*) que pueden ser fácilmente agregadas por los usuarios.

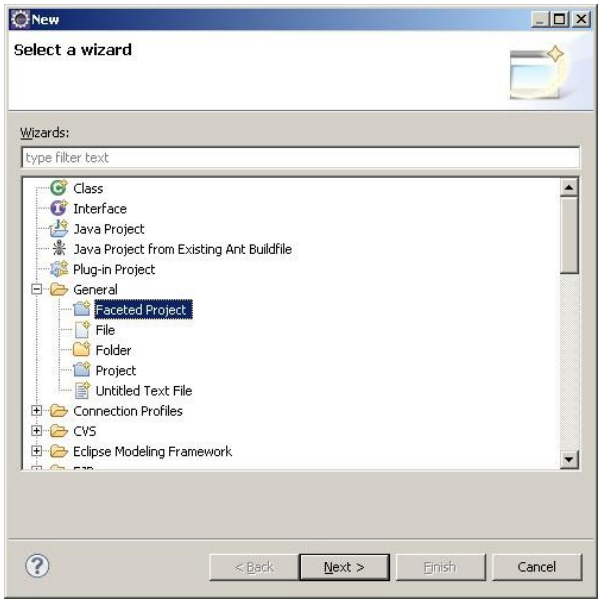

Figura 3 – Crear proyecto Faceted

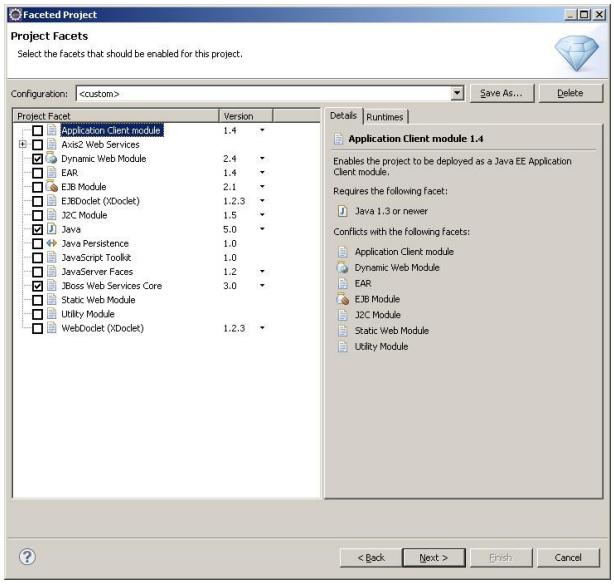

Figura 4 – Opciones Faceted

2. Configurar el *facet* Java. Se debe especificar la carpeta destino del código fuente (src) y compilado (build). Dejar valores por defecto.

3. Configurar el *facet* Dynamic Web Module. Se debe especificar el directorio de contenido Web. Dejar valores por defecto.

4. Configurar el *facet* JBossWS 3.0. Se debe especificar un Web Service Runtime. Para esto seleccionar "*New*" y completar con datos similares a los de la Figura 5. Presionar el botón "*Finish*" y luego seleccionar el *runtime* como se muestra en la Figura 6.

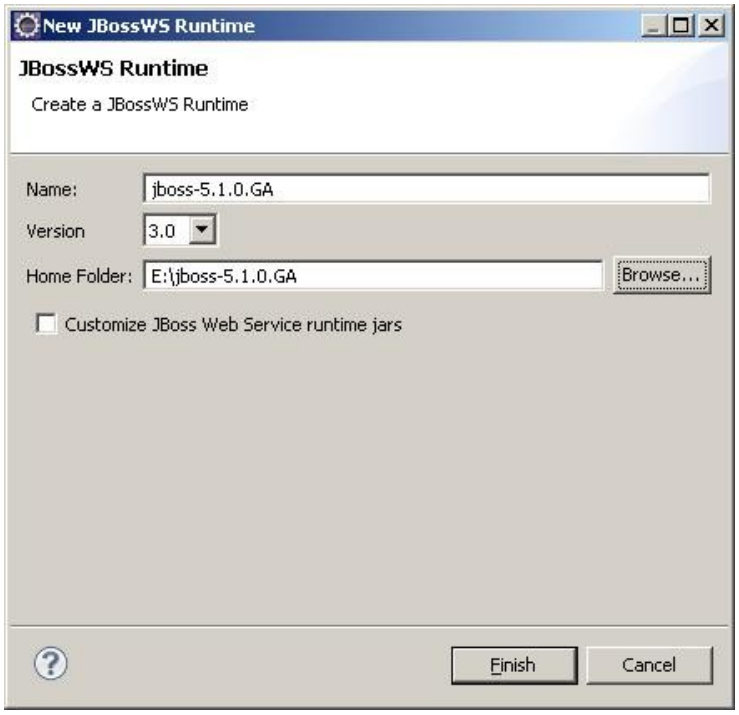

Figura 5 – Crear JBossWS Runtime

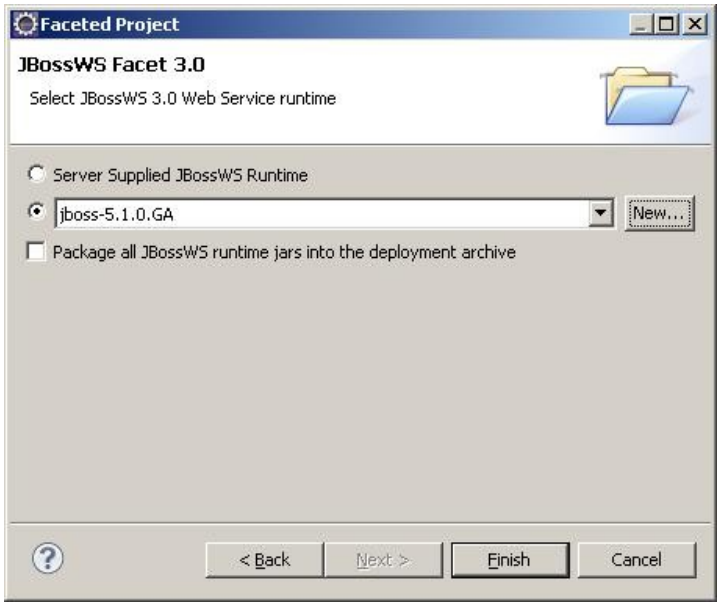

Figura 6 – Seleccionar JBossWS 3.0 Web Service Runtime

Definir un *Server Runtime* para JBoss AS

Esta configuración es necesaria para que funcione correctamente la herramienta de generación de código automática.

Los pasos a seguir para realizar esta tarea son:

1. Seleccionar del menú de Eclipse la opción Windows → Preferences.

2. Buscar la opción Server →Runtime Environments como se muestra en la Figura 7.

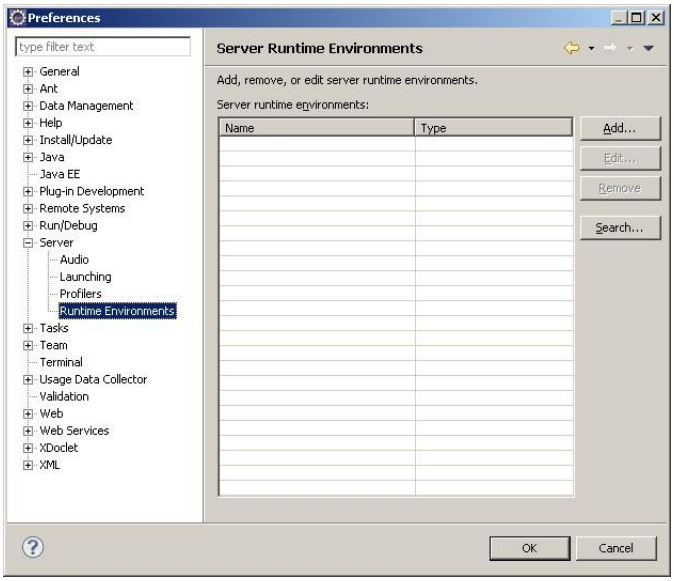

Figura 7 – Configurar JBoss Runtime

3. Seleccionar el botón *Add...* y luego la opción *JBoss Community* → *JBoss 5.1 Runtime* como se muestra en la Figura 8. Se debe alcanzar un resultado similar al de la Figura 9.

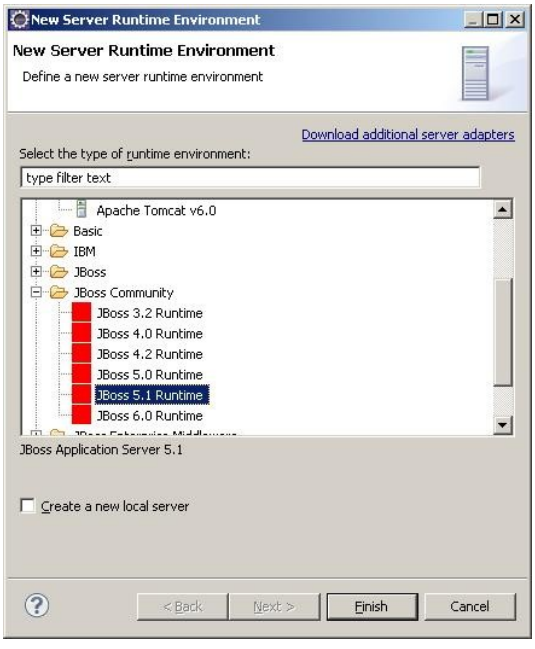

Figura 8 – Configurar JBoss Runtime parte 2

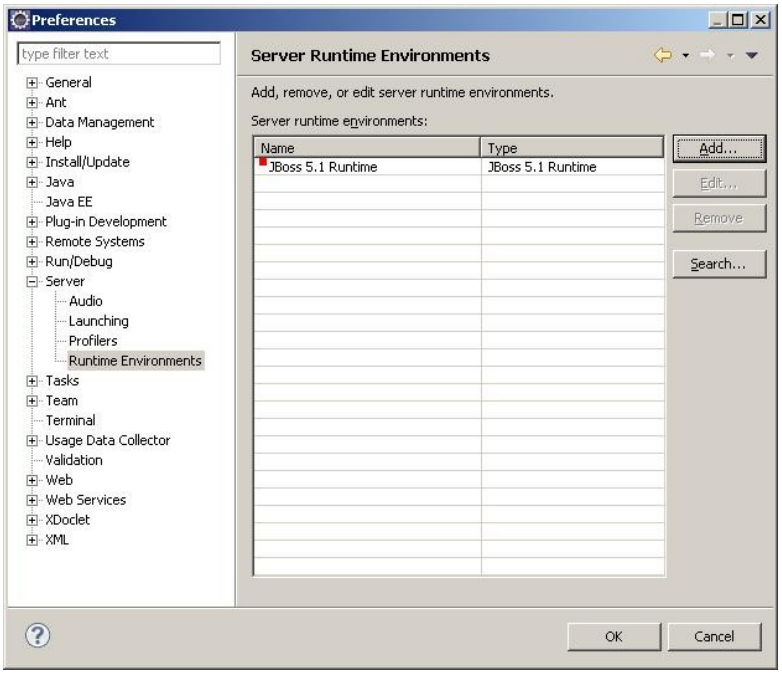

Figura 9 – Configuración completa del JBoss Runtime

Incluir Librerías y Otros Archivos Necesarios

La Aplicación Cliente requiere librerías de JBossWS y OpenSAML para su implementación. Además, requiere la Librería de Ejemplo implementada por AGESIC. Por último, es necesario también incluir en el proyecto el WSDL del servicio Certificado de Nacidos Vivos.

Los pasos a seguir para incluir estos archivos en el proyecto son:

1. Crear la carpeta lib y agregar las bibliotecas de JBossWS - Native, de OpenSAML y de AGESIC.

2. Agregar estas bibliotecas al Java Build Path del proyecto, haciendo clic derecho sobre el mismo y seleccionando *Properties* → *Java Build Path* → *Libraries* → *Add JARs…*

3. Crear la carpeta wsdl y agregar el WSDL del servicio a consumir. Este archivo debe tener extensión .wsdl para ser interpretado correctamente por Eclipse.

Nota: Las librearías requeridas y el WSDL del servicio se pueden obtener en [4].

Obtención del *token* de Seguridad emitido por la PGE

Para realizar esta tarea, se utiliza la Librearía de Ejemplo desarrollada por AGESIC. Los pasos a seguir son los siguientes:

1. Crear el *package* test.

2. Crear la clase PGEClientTest en el *package* test de forma tal que contenga un método *main* como se presenta en la Figura 10.

```
package test;

public class PGEClientTest {

	public static void main(String[] args) {

	}
}
```

Figura 10 – Clase PGEClientTest

Crear en el *main* un RSTBean especificando los datos para enviar el pedido al STS de la PGE, como se muestra en la Figura 11.

```
String userName = "Juan";
String role = "Doctor";
String service = "http://192.168.40.190:9000/Servicio";
String policyName = "urn:tokensimple";
String issuer = "BPS";

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

Figura 11 – Clase PGEClientTest

3. Como se presenta en la Figura 12, crear dos StoreBeans para almacenar los datos para acceder a los almacenes de claves que contienen los certificados y claves requeridas.

```
String alias = "alias";
String keyStoreFilePath = "c:/...";
String keyStorePwd = "password";

String trustStoreFilePath = "c:/...";
String trustStorePwd = "password";


StoreBean keyStore = new StoreBean();
keyStore.setAlias(alias);
keyStore.setStoreFilePath(keyStoreFilePath);
keyStore.setStorePwd(keyStorePwd);

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

Figura 12 – Keystore y Truststore

4. Por último, crear un PGEClient e invocar el método requestSecurityToken para obtener el *token* SAML firmado por la PGE, como se muestra en la Figura 13.

```
PGEClient client = new PGEClient();
SAMLAssertion assertionResponse =
client.requestSecurityToken(bean, keyStore, trustStore);
```

Figura 13 – Obtención del *token* SAML firmado por la PGE

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 Certificado de Nacidos Vivos, que incluya:

- información de negocio según el WSDL del servicio
- servicio y método a invocar (especificados a través de WS-Addressing)
- *token* SAML firmado por la PGE (incluido a través de WS-Security)

En este ejemplo, la invocación al servicio consta de cuatro pasos:

1. Crear las clases para consumir el servicio. A través de estas clases se crea el mensaje SOAP con la información de negocio.

2. Adjuntar en el mensaje SOAP el servicio y método a invocar.

3. Adjuntar en el mensaje SOAP el *token* SAML firmado por la PGE.

4. Consumir el servicio

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. Los pasos a seguir son los siguientes:

1. Hacer clic derecho en el archivo wsdl del servicio ubicado en la carpeta wsdl y seleccionar *Web Service* → *Generate Client* como se muestra en la Figura 14.

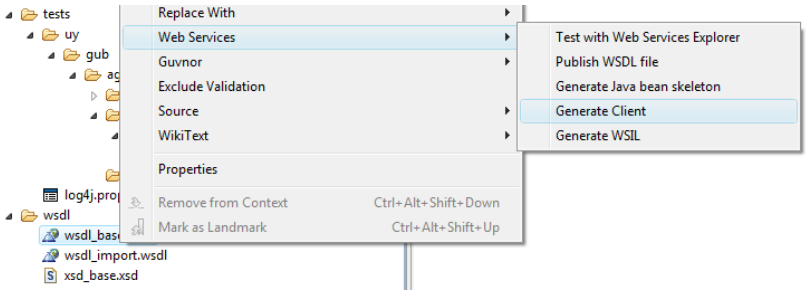

Figura 14 – Generar Clases para Consumir Web Service

2. Seleccionar JBossWS como Web Service Runtime y seleccionar el nivel de generación del cliente como "*Develop Client*", según se muestra en la Figura 15.

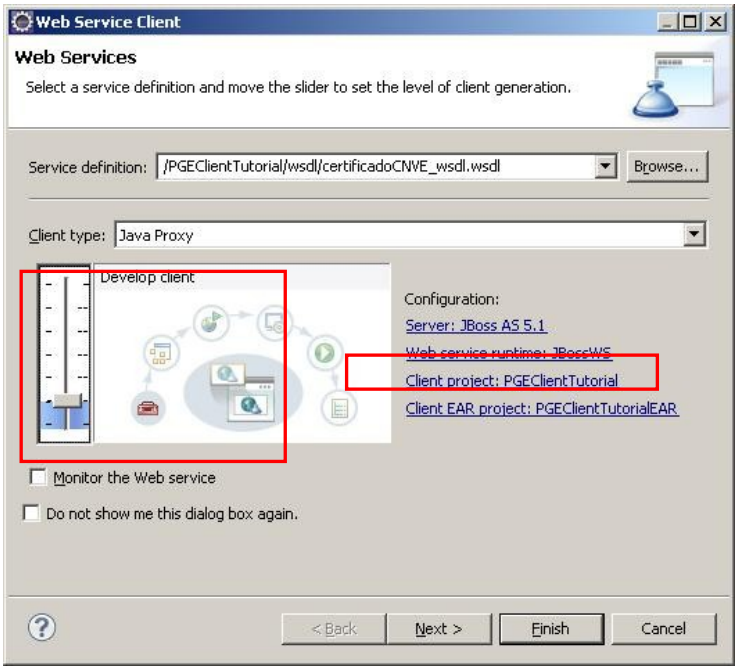

Figura 15 – Crear clases Proxy del servicio

3. Presionar "*Next*" y si se desea, modificar el nombre del paquete donde se colocan las clases generadas.

Al finalizar estos pasos, JBoss Tools genera un conjunto de clases Java, que se muestran en la Figura 16, para llevar a cabo la comunicación con el servicio de la PGE.

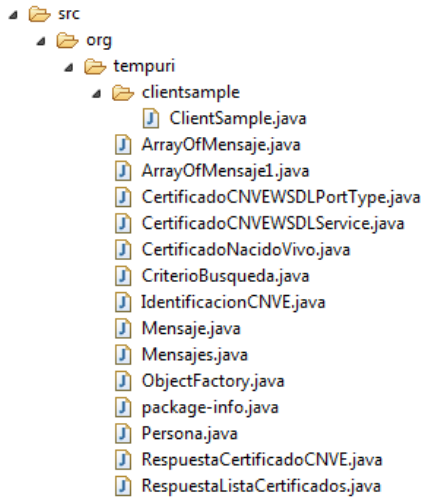

Figura 16 – Clases Generadas para Invocar al Web Service

En particular, como se muestra en la Figura 17, se genera una clase denominada ClientSample que brinda un ejemplo de invocación a los métodos del Web Service.

```
System.out.println("Create Web Service Client..");
CertificadoCNVEWSDLService service1 = new
CertificadoCNVEWSDLService();

System.out.println("Create Web Service...");
CertificadoCNVEWSDLPortType port1 =
service1.getCustomBindingCertificadoCNVEWSDLPortType();

System.out.println("Call Web Service Operation...");
System.out.println("Server said: " +
port1.registrarCNVE(null));
//Please input the parameters instead of 'null' for the
upper method!

System.out.println("Server said: " +
port1.getCertificadosByCriteria(null));
//Please input the parameters instead of 'null' for the
upper method!
```

Figura 17 – Clase ClientSample

Adjuntar en el mensaje SOAP el servicio y método a invocar.

Como se menciona previamente, la PGE requiere que en la invocación al servicio se especifique el servicio y método a invocar, utilizando los cabezales de WS-Addressing “To” y “Action”, respectivamente.

La plataforma JBoss (plataforma utilizada para el desarrollo de este tutorial) implementa las especificaciones WS-* siguiendo un diseño basado en *‘pipes & filters’* donde un mensaje SOAP pasa por una serie de *pipes* (o *handlers*) y filtros previo a su envío. Además, provee *handlers* prefabricados para poder utilizar las especificaciones WS-* minimizando al máximo las tareas de programación. Concretamente, cuenta con la clase *WSAddressingClientHandler* la cual se encarga de adjuntar los cabezales WS-Addressing al mensaje permitiendo especificarlos a través de las clases *AddressingBuilder* y *SOAPAddressingProperties*.

La Figura 18 presenta cómo utilizar este mecanismo para adjuntar los cabezales WS-Addressing requeridos por la PGE.

```
//Build handler chain
List<Handler> customHandlerChain = new
ArrayList<Handler>();
customHandlerChain.add(new WSAddressingClientHandler());


//Build addressing properties
AddressingBuilder addrBuilder =
SOAPAddressingBuilder.getAddressingBuilder();
SOAPAddressingProperties addrProps =
(SOAPAddressingProperties)addrBuilder.newAddressingProper
ties();

AttributedURI to = new AttributedURIImpl(service);
AttributedURI action = new AttributedURIImpl(method);

addrProps.setTo(to);
addrProps.setAction(action);

//Add bindings to the soap call
CertificadoCNVEWSDLService cnveService = new
CertificadoCNVEWSDLService();
CertificadoCNVEWSDLPortType port =
cnveService.getCustomBindingCertificadoCNVEWSDLPortType()
;

BindingProvider bindingProvider = (BindingProvider)port;
bindingProvider.getRequestContext().put(JAXWSAConstants.C
LIENT_ADDRESSING_PROPERTIES, addrProps);
bindingProvider.getBinding().setHandlerChain(customHandle
rChain);
```

Figura 18 – Agregar los cabezales WS-Addressing al mensaje

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 19 presenta cómo utilizar este mecanismo para adjuntar el *token* SAML requerido por la PGE.

```
//Build handler chain
...
customHandlerChain.add(new SAMLHandler());

...
//Add bindings to the soap call
...
bindingProvider.getRequestContext().put(AgesicConstants.S
AML1_PROPERTY, assertionResponse);
bindingProvider.getBinding().setHandlerChain(customHandle
rChain);
```

Figura 19 –Agregar *token* SAML al mensaje usando WS-Security

El *handler* desarrollado por AGESIC para adjuntar el *token* SAML se puede obtener en [4].

Consumir el Servicio

Por último, se debe consumir el servicio. Para esto se utiliza las clases generadas en el paso "Crear las clases para consumir el servicio", como se muestra en la Figura 20.

```
//Create input
IdentificacionCNVE idCNVE = new IdentificacionCNVE();
Persona mother = new Persona();
mother.setPrimerNombre("Marta");

CertificadoNacidoVivo solicitudCNVE = new
CertificadoNacidoVivo();
solicitudCNVE.setUsuario(userName);
solicitudCNVE.setNumeroCertificado(idCNVE);
solicitudCNVE.setDatosMadre(mother);

//Call the web service
RespuestaCertificadoCNVE response =
port.registrarCNVE(solicitudCNVE);
String code = response.getCodigoRespuesta();

System.out.println("Response code: "+code);
```

Figura 20 –Consumir el Servicio

Para ejecutar el cliente implementado, seleccionar la clase PGEClientTest, hacer clic derecho y ejecutar *Run as* → *Java Application*. La consola debería mostrar un mensaje similar al presentado en la Figura 21.

```
Codigo de respuesta: 01
```

Figura 21 – Mensaje de respuesta del servicio

Referencias

[1] Eclipse. http://www.eclipse.org/ [Accedida en Junio de 2010]

[2] JBoss Tools. http://www.jboss.org/tools [Accedida en Junio de 2010]

[3] JBoss Web Services. http://www.jboss.org/jbossws [Accedida en Junio de 2010]

[4] AGESIC. Recursos para el desarrollo de los tutoriales.

[5] keytool - Key and Certificate Management Tool. http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html [Accedida en Junio de 2010]

[6] Useful WS-Security Command-Line Tools. http://java.sun.com/webservices/docs/1.6/tutorial/doc/XWS-SecurityIntro6.html#wp526882 [Accedida en Junio de 2010]