AGESIC

Gerencia de Proyectos

Tutorial para Publicar y Consumir Novedades mediante la PGE sobre Plataforma Java

Historial de Revisiones

Fecha	Versión	Descripción	Autor	Aprobado Por
01/11/2012	1.0	Versión inicial, en tres partes independientes	Laura Rodríguez	
18/02/2013	2.0	Reescritura, unificando las tres partes	Sergio Pío Alvarez	
19/03/2013	2.1	Corrección de errores en el código fuente de ejemplo	Sergio Pío Alvarez	
08/05/2013	2.2	Ajustes por cambios en el FTP	Sergio Pio Alvarez	

agesic
agencia de gobierno electrónico y sociedad de la información

PRESIDENCIA
REPÚBLICA ORIENTAL DEL URUGUAY

Indice

agesic
agencia de gobierno electrónico
y sociedad de la información

PRESIDENCIA
REPÚBLICA ORIENTAL DEL URUGUAY

1 - Introducción

1.1 - Objetivo del documento

El objetivo de este documento es proveer información básica sobre el servicio de novedades, y a su vez servir como guía paso a paso para el desarrollo de un cliente stand-alone de la Plataforma de Gobierno Electrónico (PGE) utilizando la tecnología Java que permita publicar y consumir novedades sobre un tópico específico.

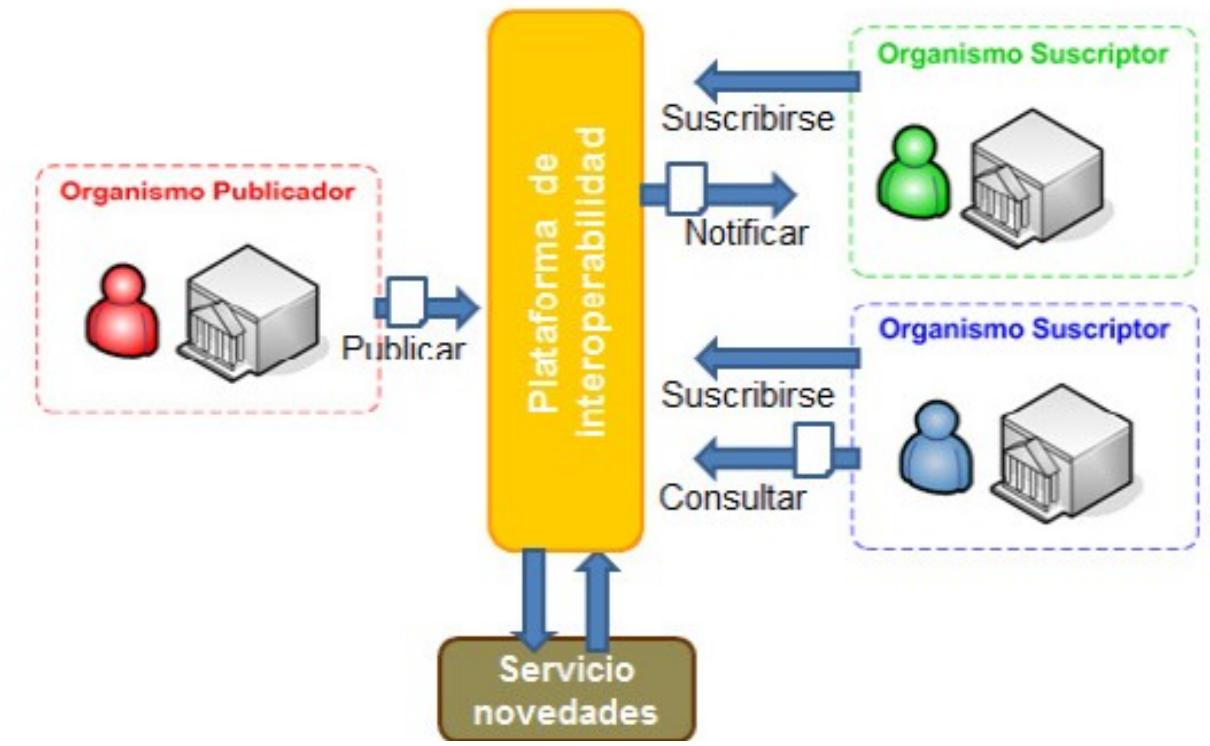
1.2 - Estructura del documento

En la sección 2 se realiza una breve descripción del sistema de novedades de la PGE, explicando el funcionamiento básico, el mecanismo de publicación de novedades y las técnicas de consumo de dichas novedades. En la sección 3 se describe el escenario de ejemplo que se implementará; en la sección 4 se ejemplifica, paso a paso, la construcción de un cliente que produzca novedades y en la sección 5 se ejemplifica la construcción de un cliente que las consuma. En la sección 6 se muestra el código fuente de ambos clientes. Finalmente, en la sección 7 se proporcionan algunas referencias para obtener mayor información.

2 - Servicio de novedades

2.1 - Descripción

El sistema de novedades pretende simplificar la interoperabilidad de los organismos usuarios de la Plataforma de Gobierno Electrónico (PGE), permitiendo que entre ellos intercambien información en forma asíncrona y confiable, mediante el mecanismo usualmente llamado Publish and Suscribe (P&S, publicación y suscripción). Bajo esta metodología de trabajo, uno o más organismos generan datos que publican en un repositorio centralizado (en la PGE), y otros organismos obtienen dichos datos a modo de novedades: cuando hay datos disponibles, los obtienen y procesan. Las novedades pueden estar referidas a múltiples dominios, por ejemplo, datos sobre personas, datos sobre importaciones y exportaciones, datos sobre asignaciones de beneficios a aspirantes, etc. A su vez, las novedades pueden estar catalogadas en tópicos, por ejemplo, las novedades de personas pueden tener un tópico dedicado a nacimientos, otro a cambios de datos personales y otro a defunciones; los tópicos son establecidos por el organismo productor.



Contexto de publicación y consumo

Los beneficios de esta forma de trabajo son múltiples:

- Asincronismo: los productores de datos y los consumidores pueden operar en forma asíncrona. Cuando un productor tiene datos para comunicar, los publica en la PGE, independientemente de si en ese mismo momento hay algún cliente esperando por dichos datos o no. A su vez, cuando un cliente requiere datos, los obtiene desde la PGE, independientemente de si el productor está disponible en el momento o no.

- Bajo acoplamiento entre productores y consumidores: lo único que deben acordar los productores y los consumidores es cuáles datos son interesantes y en qué formato se intercambian; de hecho, lo usual es que sea solo el productor quien determine cuáles datos desea compartir, y en qué formato lo hará, publicando un archivo XSD que el cliente utilizará para reconocer los datos recibidos.

- Centralización: se reduce a su mínima expresión la necesidad de que cada extremo, sea productor o consumidor, esté al tanto de qué información debe compartir con quién. Los productores definen, conjuntamente con AGESIC, cuáles datos van a compartir, y los consumidores solicitan a AGESIC el acceso a dichos datos, debiendo mantener la información en un solo lugar, en lugar de tener que hacerlo en múltiples repositorios distribuidos, controlando la disponibilidad y el acceso a cada uno por separado.

- Escalabilidad: es muy sencillo que nuevos clientes se sumen a procesar la información puesta en disponibilidad por los productores. Incluso, los productores no deben ser modificados en ninguna medida cuando nuevos clientes se presentan.

- Confiabilidad, seguridad, integridad. Todos estos beneficios son inherentes a la PGE; dado que toda la comunicación se realiza utilizando los servicios provistos por la PGE, todos sus beneficios son obtenidos al momento de utilizar el servicio de Publish and Suscribe.

2.2 - Publicación de novedades

La publicación de las novedades es hecha por los organismos productores, los cuales realizan la publicación cuando lo consideran apropiado, en el formato previamente establecido por ellos mismos. Por ejemplo, en el caso de las novedades de personas, el organismo encargado de publicar los datos informa a AGESIC cuáles datos precisamente va a publicar, proporcionando un archivo XSD que describe la estructura de tales datos (por ejemplo, de las personas, pueden publicarse su documento de identidad, nombres y apellidos, teléfonos, dirección, etc.). De esta manera, cada vez que el organismo productor tiene novedades para publicar, invoca un servicio web. Los datos publicados por los organismos productores son denominados novedades, y pueden ser de dos tipos: altas y modificaciones. A cada novedad publicada en la PGE se le asigna un identificador único secuencial; este identificador será usado posteriormente por los clientes, pero no afecta ni interesa al organismo productor.

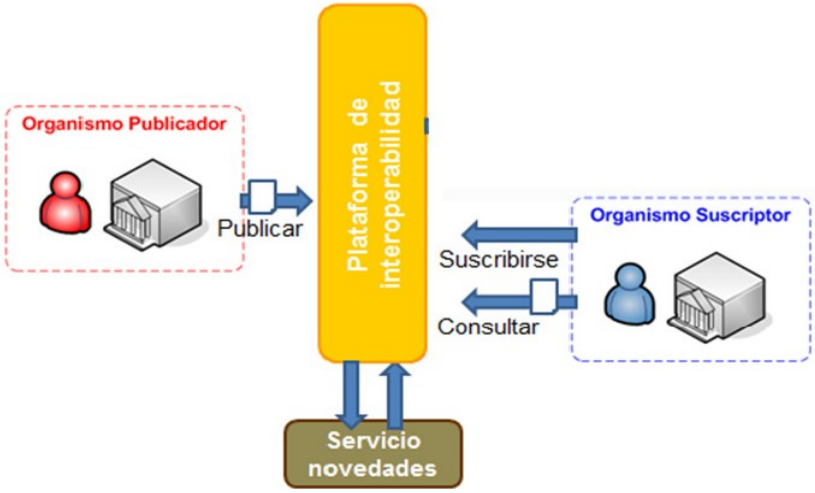
2.3 - Consumo de novedades

El consumo de las novedades por parte de los organismos consumidores puede ser realizada de dos maneras: mediante la técnica denominada pull, en la cual el cliente es el responsable de consultar a la PGE por nuevas novedades, o mediante la técnica llamada push, en la cual la PGE es responsable de enviar a los clientes previamente registrados las novedades a medida que van siendo publicadas. En ambos casos se requiere que los clientes se suscriban al servicio de novedades, para que les sea asignado un identificador único.

2.3.1 - Consumo de novedades mediante pull

El consumo mediante pull es el más sencillo para el cliente, ya que solo requiere que éste invoque en forma repetida un servicio web, identificándose a sí mismo ante la PGE (para que ésta pueda determinar quién invoca y así saber cuáles novedades ya le ha enviado). Cuando un cliente invoca este servicio web, la PGE determina cuál fue la última novedad que le ha enviado anteriormente (y el cliente ha confirmado su recepción) y procede a enviar la siguiente, si hay alguna disponible. Cuando el cliente recibe una novedad, debe tomar nota del identificador de dicha novedad, ya que en la próxima oportunidad deberá adjuntar dicho identificador en un campo

específico de la solicitud de recepción de novedad, avisando que la recibió correctamente; si no adjunta dicho identificador la siguiente vez que invoque el servicio, volverá a recibir la misma novedad.



Consumo de novedades mediante pull

Bajo esta técnica, el cliente debe invocar el servicio en forma periódica; en cada invocación, el cliente recibe una única novedad, o una respuesta vacía si no hay ninguna disponible en el momento. Cada cliente podrá realizar las sucesivas invocaciones a su tiempo, cuando lo considere apropiado, no necesitando de ningún tipo de sincronización con el productor de novedades, o con otros clientes.

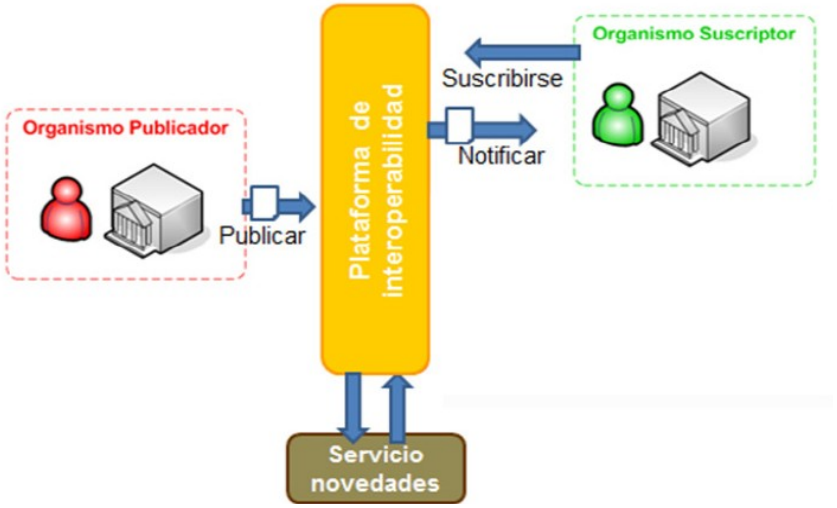
Cuando un cliente realiza una solicitud para obtener una novedad, debe enviar la siguiente información:

- Identificación de suscriptor: es un identificador otorgado por AGESIC a cada organismo cliente, de forma de que la PGE pueda identificar a cada cliente y realizar una atención personalizada a cada uno.
- Identificación del tópico: debe indicar el tópico del cual está interesado en obtener una novedad.
- ACK: debe corresponder con el identificador de la última novedad recibida correctamente. Este campo es opcional. Tener en cuenta que especificar un identificador menor que el último recibido no tiene ningún efecto (puesto que la PGE conoce el mayor efectivamente recibido para el suscriptor), mientras que enviar un identificador mayor puede tener como efecto la pérdida (no envío por parte de la PGE) de novedades. La primera vez que se consulte por una novedad no hay identificador para enviar, y dado que la PGE reconoce que no ha enviado ninguna novedad al suscriptor, envía la primera conocida; en sucesivas ocasiones, no enviar el identificador de la última novedad recibida tiene como efecto la repetición de la misma novedad (la PGE asume que la novedad enviada no fue recibida por el cliente ya que éste no confirmó su recepción).

2.3.2 - Consumo de novedades mediante push

El consumo mediante push es, en cierta forma, en sentido inverso al consumo mediante pull, ya que en lugar de ser el cliente quien invoca repetidamente un servicio web publicado en la PGE para obtener una novedad, es la PGE quien invoca un servicio web publicado por el cliente para

enviarle las novedades a medida que las va recibiendo. Este servicio debe ser implementado por el organismo consumidor de las novedades (utilizando el mismo archivo XSD provisto por el organismo productor de la mismas y acordado con AGESIC) publicado de tal forma que la PGE tenga acceso a él, y notificado a AGESIC. Bajo esta técnica, se dice que el cliente se suscribe al servicio de novedades, y las recibe directamente de la PGE. Si bien la PGE provee mecanismos de recuperación, almacenamiento y redistribución para los casos en que el servicio web de algún cliente no esté funcional cuando llega una novedad, lo deseable es que los clientes tengan disponibilidad absoluta para recibir las novedades enviadas por la PGE.



Consumo de novedades mediante push

3 - Escenario de ejemplo

3.1 - Prerrequisitos

Para lo que sigue del documento, se asume que se leyó la sección anterior, y que el lector está familiarizado con el uso e implementación de servicios web, particularmente en lenguaje Java, el uso de certificados digitales y la criptografía de clave pública. Además, se sugiere un conocimiento aunque sea básico de las especificaciones WS-Security [1], WS-Trust [2], SAML 1.1 [3].

3.2 - Requerimientos del software

La tabla de requerimientos de software presenta las herramientas y productos de *software* requeridos para desarrollar y ejecutar la Aplicación Cliente de acuerdo a este tutorial. Si bien pueden construirse clientes utilizando otras herramientas o versiones, este tutorial asume que se usan las especificadas a continuación:

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

Tabla 1 – Requerimientos de Software

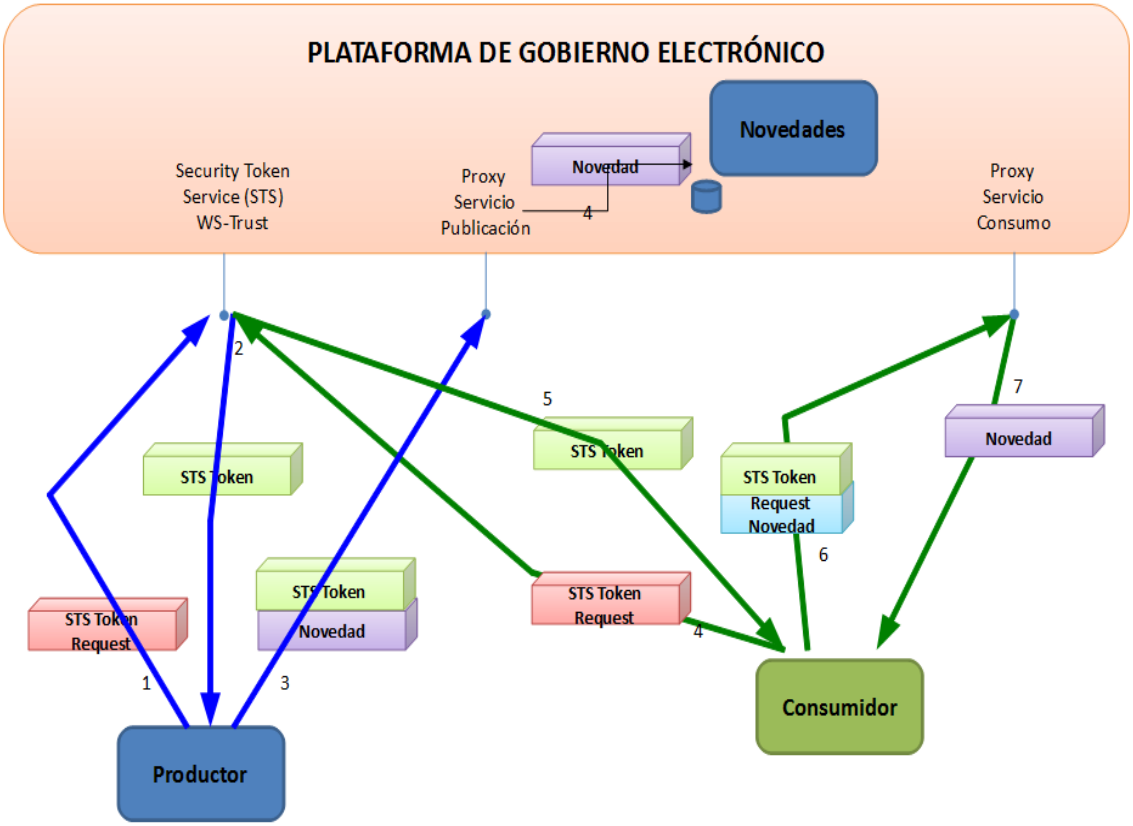
3.3 - Descripción del escenario

La siguiente figura presenta el escenario de ejemplo que se utiliza en este tutorial, en el cual intervienen dos organismos, uno como publicador de novedades y otro como consumidor de las mismas; en este caso, se . El publicador invoca un servicio en la PGE proporcionado por AGESIC, que utiliza un XSD creado por el propio proveedor, para publicar novedades; como cualquier servicio publicado en la PGE, debe obtener primero un token STS que lo autorice a invocar el servicio, siguiendo exactamente el mismo procedimiento que se utiliza para otros servicios web, y luego publicar la novedad. El organismo consumidor también debe invocar un servicio en la PGE proporcionado por AGESIC, y por tanto también necesita obtener un token STS, para luego solicitar la novedad. A nivel operativo, el servicio de novedades funciona de la siguiente manera:

1. El cliente del organismo publicador solicita un token STS, proveyendo sus credenciales, y la identificación del servicio que desea invocar; en este ejemplo, el servicio será llamado PublicarNovedades.
2. La PGE valida las credenciales proporcionadas por el cliente, identifica el servicio y si todo es correcto, emite el token STS.
3. El cliente invoca el servicio PublicarNovedades, adjuntando el token STS y los datos correspondientes a la novedad.

4. El cliente del organismo consumidor (“suscriptor”) solicita un token STS, de la misma manera que lo hizo el cliente del organismo consumidor, aunque en este caso para el servicio RecibirNovedades.
5. La PGE valida las credenciales proporcionadas por el cliente, identifica el servicio y si todo es correcto, emite el token STS.
6. El cliente invoca el servicio RecibirNovedades, adjuntando el token STS, su identificador de suscriptor y el identificador del tópico del cual desea consumir la novedad.
7. Si los datos son correctos, la PGE entrega al cliente la siguiente novedad no entregada a dicho cliente.

Estos pasos pueden visualizarse gráficamente en la siguiente imagen:



Secuencia de publicación y consumo de novedades

3.4 - Descripción de las novedades

Para este tutorial, se utilizará un sistema de novedades de ejemplo, relativo a personas con un conjunto reducido de datos (cédula de identidad, nombre, apellido y fecha de nacimiento), bajo un tópico conocido como “NoticiasAgesic”. Esto está representado por el siguiente XSD (adjunto a este documento debería encontrase un archivo llamado noticias.xsd con este mismo contenido):

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://noticias.agesic.gub.uy/"
 xmlns:tns="http://noticias.agesic.gub.uy/" elementFormDefault="qualified">
 <element name='noticia' type="tns:noticiaType" />
   <complexType name='noticiaType'>
     <sequence>
       <element minOccurs='0' name='personas'>
         <complexType>
           <sequence>
             <element maxOccurs='unbounded' minOccurs='0' name='persona'
type="tns:personaType" />
           </sequence>
         </complexType>
       </element>
     </sequence>
   </complexType>
   <complexType name="personaType">
     <sequence>
       <element name="nombre" type="string"></element>
       <element name="apellido" type="string"></element>
       <element name="fechaNacimiento" type="date"></element>
       <element name="ci" type="string"></element>
     </sequence>
   </complexType>
</schema>
```

Este xsd se utiliza como parte del wsdl del servicio que permite publicar novedades en la PGE.

3.5 - Datos de los servicios y credenciales de acceso

A continuación se muestran en forma tabulada los datos correspondientes a los servicios involucrados, y las credenciales que serán usadas por ellos para poder invocarlos.

Servicio PublicarNovedades		
Dato	**Valor**	**Comentarios**
Nombre de Usuario	User	Este dato es solo con fines de auditoría, y no es validado.
Rol de Usuario	ou=capacitacion, o=agesic	Este dato debe ser solicitado por quien publica novedades a AGESIC.
URL física	https://testservicios.pge.red.uy:6081/NoticiasService/NuevaNovedadService	Dirección física del servicio en la PGE.
URL lógica	http://test_servicios.pge.red.uy/agesic/servicioPublicacionNovedadesTest	Este dato será proporcionado por AGESIC y corresponde a la URL que identifica el servicio que se desea invocar. Se corresponde con wsa:To.
Método del Servicio	nuevaNovedad	Este parámetro identifica el método que se desea invocar, dentro del servicio web. Debe especificarse este valor para ser enviado en el campo wsa:Action.
PolicyName	urn:tokensimple	El tipo de token; en testing es siempre urn:tokensimple, en producción es siempre urn:std15.
Tópico	NoticiasAgesic	Este valor relaciona la novedad que se está enviando con el tópico dentro del sistema de novedades. Este valor es asignado por AGESIC al dar de alta en el sistema la publicación de novedades “NoticiasAgesic”.
Productor	AGESIC	El productor es quien tiene permisos en el sistema

		de novedades para producir novedades con ese tópico.

Tabla 2 – Datos para la Implementación del Escenario

Servicio ConsumirNovedades		
Dato	**Valor**	**Comentarios**
Nombre de Usuario	User	Este dato es solo con fines de auditoría, y no es validado.
Rol de Usuario	ou=TEST_SUBSCRIBE_PULL, o=TEST_PE	Este dato debe ser solicitado por quien consume novedades a AGESIC.
URL física	https://testservicios.pge.red.uy:6082/novedades/RecibirNovedadService	Dirección física del servicio en la PGE.
URL lógica	http://test_servicios.pge.red.uy/agesic/servicioRecepcionNovedadesPullTest	Este dato será proporcionado por AGESIC y corresponde a la URL que identifica el servicio que se desea invocar. Se corresponde con wsa:To.
Método del Servicio	recibirNovedad	Este parámetro identifica el método que se desea invocar, dentro del servicio web. Debe especificarse este valor para ser enviado en el campo wsa:Action.
PolicyName	urn:tokensimple	El tipo de token; en testing es siempre urn:tokensimple, en producción es siempre urn:std15.
Tópico	NoticiasAgesic	Este valor relaciona la novedad que estoy enviando con el tópico dentro del sistema de novedades. Este valor es proporcionado por AGESIC al dar de alta en el sistema la publicación de novedades “Noticias Agesic”.
Suscriptor	Testing	El suscriptor es quien tiene permisos en el sistema de novedades para consumir novedades con ese tópico y en modalidad PULL.

Los datos de negocio a incluir en las invocaciones están especificados en los archivos que contienen las descripciones de los respectivos servicios (WSDLs). En ellos también se incluyen la URLs de los servicios donde el cliente debe enviar los mensajes SOAP para invocarlos (valor del atributo location del tag soap:address, dentro del tag wsdl:service); en testing, estas URL deben comenzar con “https://testservicios.pge.red.uy” (la URL de la PGE). En este ejemplo las URLs de los servicios de novedades de Noticias de Agesic son https://testservicios.pge.red.uy:6081/NoticiasService/NuevaNovedadService, que se encuentra especificada en el archivo NuevaNovedadService.wsdl, y https://testservicios.pge.red.uy:6082/novedades/RecibirNovedadService que se encuentra especificada en el archivo RecibirNovedadService.wsdl.

4 - Publicación de novedades

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

La implementación del escenario comprenderá las siguientes etapas:

- Obtener los materiales necesarios: librerías, certificados, wsdl, etc.
- Crear un proyecto Java Faceted en el entorno Eclipse IDE.
- Implementación del cliente
- Invocación del Servicio

En las siguientes secciones se describen en detalle cada una de ellas.

4.1 - Obtener los materiales necesarios

Como primer paso se deben obtener los materiales necesarios. Esto incluye las librerías adicionales que deberán ser agregadas a los clientes, los keystores conteniendo los certificados digitales, y el WSDL que define el servicio que se invocará. Esto puede hacerse desde el servidor FTP de AGESIC, en la siguiente URL: ftp://ftp.agesic.gub.uy/Tutoriales/Java/PublishAndSuscribe/ (usuario: agesic; contraseña: publico).

Los archivos NuevaNovedadService.wsdl y noticias.xsd corresponden a la definición del servicio que se debe invocar para publicar las novedades. Los archivos con extensión keystore y truststore contienen los certificados digitales que permiten invocar servicios web en la PGE, incluyendo el que otorga el token STS. La carpeta lib contiene un conjunto de archivos jar, de los cuales algunos son necesarios agregar al proyecto para la compilación, y otros agregar al servidor JBoss AS para la ejecución.

4.2 - Crear y configurar el proyecto Java en Eclipse IDE

4.2.1 - Crear el proyecto

Para poder crear un cliente de un servicio web utilizando Eclipse IDE, es necesario utilizar algunas herramientas que solo están disponibles en las aplicaciones web; por esta razón, el proyecto que contendrá al cliente del servicio web debe ser un proyecto Java Faceted, como se explica a continuación:

1. Abrir el Eclipse IDE.
2. Seleccionar *File → New → Other → General → Faceted Project,* crear un nuevo proyecto con el nombre Publicar_*Novedades_PGE* y seleccionar los facets ***Dynamic Web Module 2.4 , Java 6.0 y JBoss Web Service Core 3.0***.
3. Hacer click en Next.
4. Configurar la carpeta destino del código fuente (src) y compilado (build), así como también el directorio de contenido Web.
5. Seleccionar el JBossWS Runtime y presionar el botón Finish.

4.2.2 - Incluir librerías y otros archivos necesarios

La Aplicación Cliente requiere librerías de JBossWS y OpenSAML, así como la Librería PGEClient-1.6.jar implementada por AGESIC. Para incluir estos archivos en el proyecto, realizar el siguiente procedimiento:

1. Hacer clic derecho sobre el nombre del proyecto, seleccionar *New* → *Folder* y crear la carpeta de nombre *lib*. Copiar en dicha carpeta **todos** los archivos que se encuentran en la carpeta lib de materiales.
2. Agregar todas las bibliotecas copiadas en el paso anterior al Java Build Path del proyecto, haciendo clic derecho sobre el proyecto y luego seleccionar *Properties* → *Java Build Path* → *Libraries* → *Add JARs…;* en la ventana que se despliega, seleccionar los archivos y hacer clic en el botón *Open*.
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*.

Luego, dejar disponibles la definición del servicio web y los keystores, para ser referenciados posteriormente desde la aplicación:

1. Crear, dentro de Eclipse, una carpeta denominada wsdl y agregar todos los archivos de la carpeta wsdl de materiales de AGESIC.
2. Crear, también dentro de Eclipse, una carpeta denominada keystores y copiar todos los archivos de la carpeta keystores de materiales de AGESIC.

4.3 - Invocación del Servicio

La invocación del servicio de publicación de novedades consta de los siguientes pasos:

1. Crear las clases (stubs) para consumir el servicio, a partir de su definición (WSDL y XSD). A través de estas clases se crearán los mensajes SOAP con la información de negocio.
2. Implementar la solicitud del token SAML firmado por PGE.
3. Adjuntar al servicio la información de direccionamiento (addressing) y el token STS.
4. Configurar la información sobre el productor y tópico.
5. Consumir el servicio.
6. Procesar la respuesta.

4.3.1 - Crear las clases para consumir el servicio

Para esta tarea se utilizará 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 NuevaNovedadService.wsdl ubicado en la carpeta wsdl y seleccionar *Web Service* → *Generate Client*.
2. Seleccionar *JBossWS* como *Web Service Runtime* y seleccionar el nivel de generación del cliente como "*Develop Client*".

3. Presionar “Next” y si se desea, modificar el nombre del paquete donde se colocarán las clases generadas (en lo que resta del tutorial se asumirá que se mantiene el nombre sugerido; si se modifica, habrá que hacer los ajustes necesarios).

4.3.2 - Obtención del token de Seguridad emitido por la PGE

Para realizar esta tarea, se utilizará la biblioteca PGEClient-1.6.jar, la cual fue desarrollada por AGESIC para simplificar la invocación de servicios web a través de la PGE. Los pasos a seguir son los siguientes:

1. En la clase generada como *ClientSample.java* dentro del paquete *uy.gub.agesic.novedades* importar las clases a utilizar.

```
import uy.gub.agesic.ps.samples.novedad.enviarnovedad.*;
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;
```

2. Colocar en el método *main* (antes del código autogenerado) la información para realizar la solicitud del token STS a la PGE:

```
String userName = "user";
String role = "ou= capacitacion, o=agesic";
String service =
"http://test_servicios.pge.red.uy/agesic/servicioPublicacionNovedadesTest";
String policyName = "urn:tokensimple";
String issuer = "AGESIC";
```

3. Luego crear un RSTBean especificando los datos para enviar la solicitud al servidor STS de la PGE. En el pedido se debe incluir 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 debe especificar la dirección del STS, la cual en el ambiente de testing es siempre *https://testservicios.pge.red.uy:6051/TrustServer/SecurityTokenServiceProtected.*

```
//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/SecurityTokenServiceProtected";
```

4. Crear dos *StoreBeans*, los cuales se utilizarán para almacenar los datos de acceso a los almacenes de claves que contienen los certificados y claves requeridas (keystore con el certificado de persona jurídica y de SSL, y truststore con el certificado de la plataforma testservicios.pge.red.uy). Las rutas que se especifican en las variables *keyStoreFilePath* y *trustStoreFilePath* apuntan a los archivos *agesic_v4.0.keystore* y *agesic_v2.0.truststore*

respectivamente que se encuentran ubicados en la carpeta keystores recientemente creada.

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

String keyStoreFilePath="<path>/keystores/agesic_v4.0.keystore";
String keyStorePwd = "agesic";

String trustStoreFilePath = "<path>/keystores/agesic_v2.0.truststore";
String trustStorePwd = "agesic";

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

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

Nota: el texto “<path>” debe ser sustituido por la ruta al proyecto.

5. Para que la invocación al servicio STS y al servicio final puedan efectuarse vía HTTPS, deberán configurarse ciertas propiedades SSL en el contexto de la invocación. Estas propiedades harán referencia a los almacenes de claves que se configuraron anteriormente (si no se realiza esto, Java intentará utilizar una configuración por defecto, con otros keystores y truststores, y por tanto la invocación fallará).

```
System.setProperty("javax.net.ssl.keyStore", keyStoreFilePath);
System.setProperty("javax.net.ssl.keyStorePassword",keyStorePwd);
System.setProperty("javax.net.ssl.trustStore",trustStoreFilePath);
System.setProperty("javax.net.ssl.trustStorePassword",trustStorePwd);
```

6. Crear una instancia de la clase *uy.gub.agesic.sts.client.PGEClient* e invocar el método *requestSecurityToken* para obtener el *token* SAML firmado por la PGE. Este token será adjuntando posteriormente en la invocación al servicio final como prueba de estar autenticado (no es prueba de estar autorizado a invocar el servicio).

```
//No olvidar realizar el import de la clase uy.gub.agesic.sts.client.PGEClient

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, keyStore, trustStore, stsUrl);
  String stringRepresentation = assertionResponse.toString();
} catch (Exception e) {
  e.printStackTrace();
  System.exit(1);
}
```

Nota importante: se debe verificar que la hora del servidor STS se encuentre sincronizada con la hora actual de su PC (incluyendo segundos). En caso de ser necesario, atrasar o adelantar la hora del PC unos minutos. Si se tiene desactualizada la hora, ocurrirá un error en la ejecución.

4.3.3 - Obtener el puerto de acceso

Para poder invocar el servicio, se requiere un puerto de acceso al mismo. Este puerto está dado por la clase NuevaNovedadService, que fue creada durante la importación del WSDL al proyecto. El siguiente código muestra cómo obtener una referencia a esta clase:

```
NuevaNovedadService_Service service1 = new NuevaNovedadService_Service();
NuevaNovedadService port = service1.getNuevaNovedadPort();
```

4.3.4 - Configurar WSAddressing y WSSecurity

Como se mencionó anteriormente, la PGE requiere que en la invocación al servicio se especifique el identificador del servicio y del método a invocar. Para esto, se utilizan los cabezales de WS-Addressing “To” y “Action”, respectivamente. Para saber cómo determinar los valores para estos cabezales, ver la sección Apéndice 3 – Determinar el valor del atributo soap:Action para un servicio. En este ejemplo, los valores a especificar son “http://test_servicios.pge.red.uy/agesic/servicioPublicacionNovedadesTest” para el cabezal “To” y “nuevaNovedad” para el cabezal “Action”, como muestra la tabla 2.

Para adjuntar el token SAML, se utiliza un manejador (handler) proporcionado por AGESIC. Este manejador requiere que se ponga en el contexto de la sesión el token SAML obtenido en el paso anterior, bajo el nombre correspondiente a la constante *AgesicConstants.SAML1_PROPERTY*.

Finalmente, se deben instalar los manejadores de WSAddressing y WSSecurity. Esto consiste en agregar a la lista de manejadores corrientes, una instancia de cada una de las siguientes clases:

- org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler
- org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer
- uy.gub.agesic.jbossws.SAMLHandler

Todos estos pasos se ilustran en el siguiente código:

```
//No olvidar realizar los imports necesarios:
//import java.util.ArrayList;
//import java.util.List;
//import java.util.Map;
//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.extensions.addressing.AttributedURIImpl;
//import org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler;
//import org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer;
//import uy.gub.agesic.AgesicConstants;
//import uy.gub.agesic.jbossws.SAMLHandler;

//Construir las propiedades de WSAddressing
AddressingBuilder addrBuilder = SOAPAddressingBuilder.getAddressingBuilder();
SOAPAddressingProperties addrProps =
 (SOAPAddressingProperties)addrBuilder.newAddressingProperties();
String actionStr = "nuevaNovedad";
AttributedURI soapTo = new AttributedURIImpl(service);
AttributedURI soapAction = new AttributedURIImpl(actionStr);
addrProps.setTo(soapTo);
addrProps.setAction(soapAction);

//Obtener el mapa de propiedades de sesión
BindingProvider bindingProvider = (BindingProvider)port;
Map<String, Object> reqContext = bindingProvider.getRequestContext();

//Agregar las propiedades de WSAddressing
reqContext.put(JAXWSAConstants.CLIENT_ADDRESSING_PROPERTIES, addrProps);

//Agregar las propiedades de WSSecurity
reqContext.put(AgesicConstants.SAML1_PROPERTY, assertionResponse);

//Instalar los manejadores de WSAddressing y WSSecurity, y el manejador SAML
List<Handler> handlerChain = new ArrayList<Handler>();
handlerChain.add(new WSAddressingClientHandler());
handlerChain.add(new WSSecurityHandlerServer());
handlerChain.add(new SAMLHandler());
//Aquí habría que instalar la lista de maneadores, pero como será necesario
//agregar un manejar más, se deja para más adelante
//bindingProvider.getBinding().setHandlerChain(customHandlerChain);
```

4.3.5 - Definir productor y tópico

Hasta aquí, el código implementado no es diferente del requerido para invocar cualquier otro servicio expuesto a través de la PGE. Lo que sigue es específico de la publicación de novedades.

A continuación se debe especificar los datos correspondientes al productor (identificar quién es el que está publicando la novedad) y el tópico (dónde desea publicar la novedad). Los datos de productor y tópico asociados a la novedad también deben colocarse en el cabezal SOAP, para lo cual se utiliza otro manejador proporcionado por AGESIC, llamado *PublishSubscriberHeadersHandler*, que se encuentra dentro del paquete *uy.gub.agesic.novedades*, de la misma manera que se instalaron los manejadores en el paso previo.

```
String productor = "AGESIC";
String topico = "NoticiasAgesic";

PublishSubscribeHeadersHandler handler =
   new PublishSubscribeHeadersHandler(productor, topico);
handlerChain.add(handler);

//Ahora sí instalar la lista de manejadores
bindingProvider.getBinding().setHandlerChain(handlerChain);
```

4.3.6 - Consumir el Servicio

Finalmente, después de haber realizado toda la configuración necesaria, se puede proceder a consumir el servicio. Luego de hacerlo, se puede obtener el id de la notificación, a los efectos de reconocer una posible respuesta.

```
PersonaType persona = new PersonaType();
persona.setApellido("ApellidoPersona");
persona.setCi("DocumentoPersona");
persona.setNombre("NombrePersona");

NoticiaType noticia = new NoticiaType();
noticia.setPersonas(new NoticiaType.Personas());
noticia.getPersonas().getPersona().add(persona);

port.nuevaNovedad(noticia);
String nofitificacionId = handler.getNotificationId();

System.out.println("Novedad Enviada! ID Notificacion: "+nofitificacionId);
```

Para ejecutar el cliente implementado hacer clic derecho sobre la clase *ClientSample* y seleccionar la opción *Run as → Java Application*. La consola debería mostrar un mensaje similar al presentado:

```
Novedad Enviada! ID Notificacion: XXXX
```

En el Apéndice 1 – Publicar Novedad: código fuente completo se incluye el código fuente completo del cliente que publica novedades.

5 - Consumo de novedades

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

La implementación del escenario comprenderá las siguientes etapas:

- Obtener los materiales necesarios: librerías, certificados, wsdl, etc.
- Crear un proyecto Java Faceted en el entorno Eclipse IDE.
- Implementación del cliente
- Invocación del Servicio

En las siguientes secciones se describen en detalle cada una de ellas.

5.1 - Obtener los materiales necesarios

Como primer paso se deben obtener 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. Esto puede hacerse desde el servidor FTP de AGESIC, en la siguiente URL: ftp://ftp.agesic.gub.uy/Tutoriales/Java/PublishAndSuscribe/ (usuario: agesic; contraseña: publico). Los archivos RecibirNovedadService.wsdl y noticias.xsd corresponden a la definición del servicio que se debe invocar para recibir las novedades. Los archivos con extensión keystore y truststore contienen los certificados digitales que permiten invocar servicios web en la PGE, incluyendo el que otorga el token STS. La carpeta lib contiene un conjunto de archivos jar, de los cuales algunos son necesarios agregar al proyecto para la compilación, y otros agregar al servidor JBoss AS para la ejecución.

5.2 - Crear un proyecto Java en Eclipse IDE

5.2.1 - Crear el proyecto

Para poder crear un cliente de un servicio web utilizando Eclipse IDE, es necesario utilizar algunas herramientas que solo están disponibles en las aplicaciones web; por esta razón, el proyecto que contendrá al cliente del servicio web debe ser un proyecto Java Faceted, como se explica a continuación:

1. Abrir el Eclipse IDE.
2. Seleccionar *File → New → Other → General → Faceted Project,* crear un nuevo proyecto con el nombre *Recibir_Novedades_PGE* y seleccionar los facets ***Dynamic Web Module 2.4 , Java 6.0 y JBoss Web Service Core 3.0***.
3. Hacer click en Next.
4. Configurar la carpeta destino del código fuente (src) y compilado (build), así como también el directorio de contenido Web.
5. Seleccionar el JBossWS Runtime y presionar el botón Finish.

5.2.2 - Incluir Librerías y Otros Archivos Necesarios

La Aplicación Cliente requiere librerías de JBossWS y OpenSAML, así como la Librería PGEClient-1.6.jar implementada por AGESIC. Para incluir estos archivos en el proyecto, realizar el siguiente procedimiento:

1. Hacer clic derecho sobre el nombre del proyecto, seleccionar *New* → *Folder* y crear la carpeta de nombre *lib*. Copiar en dicha carpeta **todos** los archivos que se encuentran en la carpeta lib de materiales.
2. Agregar todas las bibliotecas copiadas en el paso anterior al Java Build Path del proyecto, haciendo clic derecho sobre el proyecto y luego seleccionar *Properties* → *Java Build Path* → *Libraries* → *Add JARs…;* en la ventana que se despliega, seleccionar los archivos y hacer clic en el botón *Open*.
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*.

Luego, dejar disponibles la definición del servicio web y los keystores, para ser referenciados posteriormente desde la aplicación:

1. Crear, dentro de Eclipse, una carpeta denominada wsdl y agregar todos los archivos de la carpeta wsdl de materiales de AGESIC.
2. Crear, también dentro de Eclipse, una carpeta denominada keystores y copiar todos los archivos de la carpeta keystores de materiales de AGESIC.

5.3 - Invocación del Servicio

La invocación del servicio de publicación de novedades consta de los siguientes pasos:

1. Crear las clases (stubs) para consumir el servicio, a partir de su definición (WSDL y XSD). A través de estas clases se crearán los mensajes SOAP con la información de negocio.
2. Implementar la solicitud del token SAML firmado por PGE.
3. Adjuntar al servicio la información de direccionamiento (addressing) y el token STS.
4. Configurar la información sobre el productor y tópico.
5. Consumir el servicio.
6. Procesar la respuesta.

5.3.1 - Crear las clases para consumir el servicio

Para esta tarea se utilizará 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 RecibirNovedadService.wsdl ubicado en la carpeta wsdl y seleccionar *Web Service* → *Generate Client*.
2. Seleccionar *JBossWS* como *Web Service Runtime* y seleccionar el nivel de generación del cliente como “*Develop Client”*.

3. Presionar “Next” y si se desea, modificar el nombre del paquete donde se colocarán las clases generadas (en lo que resta del tutorial se asumirá que se mantiene el nombre sugerido; si se modifica, habrá que hacer los ajustes necesarios).

5.3.2 - Obtención del token de Seguridad emitido por la PGE

Para realizar esta tarea, se utilizará la biblioteca PGEClient-1.6.jar, la cual fue desarrollada por AGESIC para simplificar la invocación de servicios web a través de la PGE. Los pasos a seguir son los siguientes:

1. En la clase generada como *ClientSample.java* dentro del paquete *uy.gub.agesic.ps.samples.novedad.recibirnovedad.clientsample* importar las clases a utilizar.

```
import uy.gub.agesic.ps.samples.novedad.reibirnovedad.*;
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;
```

2. Colocar en el método *main* (antes del código autogenerado) la información para realizar la solicitud del token STS a la PGE:

```
String userName = "user";
String role = "ou=TEST_SUBSCRIBE_PULL,O=TEST_PE";
String service =
  "http://test_servicios.pge.red.uy/agesic/servicioRecepcionNovedadesPullTest";
String policyName = "urn:tokensimple";
String issuer = "AGESIC";
```

3. Luego crear un RSTBean especificando los datos para enviar la solicitud al servidor STS de la PGE. En el pedido se debe incluir 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 debe especificar la dirección del STS, la cual en el ambiente de testing es siempre *https://testservicios.pge.red.uy:6051/TrustServer/SecurityTokenServiceProtected.*

```
//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/SecurityTokenServiceProtected";
```

4. Crear dos *StoreBeans*, los cuales se utilizarán para almacenar los datos de acceso a los almacenes de claves que contienen los certificados y claves requeridas (keystore con el certificado de persona jurídica y de SSL, y truststore con el certificado de la plataforma testservicios.pge.red.uy). Las rutas que se especifican en las variables *keyStoreFilePath* y *trustStoreFilePath* apuntan a los archivos *agesic_v4.0.keystore* y *agesic_v2.0.truststore*

respectivamente que se encuentran ubicados en la carpeta keystores recientemente creada.

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

String keyStoreFilePath="<path>/keystores/agesic_v4.0.keystore";
String keyStorePwd = "agesic";

String trustStoreFilePath = "<path>/keystores/agesic_v2.0.truststore";
String trustStorePwd = "agesic";

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

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

Nota: el texto “<path>” debe ser sustituido por la ruta al proyecto.

5. Para que la invocación al servicio STS y al servicio final puedan efectuarse vía HTTPs, deberán configurarse ciertas propiedades SSL en el contexto de la invocación. Estas propiedades harán referencia a los almacenes de claves que se configuraron anteriormente (si no se realiza esto, Java intentará utilizar una configuración por defecto, con otros keystores y truststores, y por tanto la invocación fallará)

```
System.setProperty("javax.net.ssl.keyStore", keyStoreFilePath);
System.setProperty("javax.net.ssl.keyStorePassword",keyStorePwd);
System.setProperty("javax.net.ssl.trustStore",trustStoreFilePath);
System.setProperty("javax.net.ssl.trustStorePassword",trustStorePwd);
```

6. Crear una instancia de la clase *uy.gub.agesic.sts.client.PGEClient* e invocar el método *requestSecurityToken* para obtener el *token* SAML firmado por la PGE. Este token será adjuntando posteriormente en la invocación al servicio final, como prueba de estar autenticado (no es prueba de estar autorizado a invocar el servicio).

```
//No olvidar realizar el import de la clase uy.gub.agesic.sts.client.PGEClient

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, keyStore, trustStore, stsUrl);
  System.out.println(stringRepresentation);
} catch (Exception e) {
  e.printStackTrace();
  System.exit(1);
}
```

Nota importante: se debe verificar que la hora del servidor STS se encuentre sincronizada con la hora actual de su PC (incluyendo segundos). En caso de ser necesario, atrasar o adelantar la hora del PC unos minutos. Si se tiene desactualizada la hora, ocurrirá un error en la ejecución.

5.3.3 - Obtener el puerto de acceso

Para poder invocar el servicio, se requiere un puerto de acceso al mismo. Este puerto está dado por la clase NuevaNovedadService, que fue creada durante la importación del WSDL al proyecto. El siguiente código muestra cómo obtener una referencia a esta clase:

```
RecibirNovedadService_Service service1 = new RecibirNovedadService_Service();
RecibirNovedadService port = service1.getRecibirNovedadPort();
```

5.3.4 - Configurar WSAddressing y WSSecurity

Como se mencionó anteriormente, la PGE requiere que en la invocación al servicio se especifique el identificador del servicio y del método a invocar. Para esto, se utilizan los cabezales de WS-Addressing “To” y “Action”, respectivamente. Para saber cómo determinar los valores para estos cabezales, ver la sección Apéndice 3 – Determinar el valor del atributo soap:Action para un servicio. En este ejemplo, los valores a especificar son “http://test_servicios.pge.red.uy/agesic/servicioRecepcionNovedadesPullTest” para el cabezal “To” y “recibirNovedad” para el cabezal “Action”, como muestra la tabla 2.

Para adjuntar el token SAML, se utiliza un manejador (handler) proporcionado por AGESIC. Este manejador requiere que se ponga en el contexto de la sesión el token SAML obtenido en el paso anterior, bajo el nombre correspondiente a la constante *AgesicConstants.SAML1_PROPERTY*.

Finalmente, se deben instalar los manejadores de WSAddressing y WSSecurity. Esto consiste en agregar a la lista de manejadores corrientes, una instancia de cada una de las siguientes clases:

- org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler
- org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer
- uy.gub.agesic.jbossws.SAMLHandler

Todos estos pasos se ilustran en el siguiente código:

```
//No olvidar realizar los imports necesarios:
//import java.util.ArrayList;
//import java.util.List;
//import java.util.Map;
//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.extensions.addressing.AttributedURIImpl;
//import org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler;
//import org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer;
//import uy.gub.agesic.AgesicConstants;
//import uy.gub.agesic.jbossws.SAMLHandler;

//Construir las propiedades de WSAddressing
AddressingBuilder addrBuilder = SOAPAddressingBuilder.getAddressingBuilder();
SOAPAddressingProperties addrProps =
  (SOAPAddressingProperties)addrBuilder.newAddressingProperties();
String actionStr = "recibirNovedad";
```

```
AttributedURI soapTo = new AttributedURIImpl(service);
AttributedURI soapAction = new AttributedURIImpl(actionStr);
addrProps.setTo(soapTo);
addrProps.setAction(soapAction);

//Obtener el mapa de propiedades de sesión
BindingProvider bindingProvider = (BindingProvider)port;
Map<String, Object> reqContext = bindingProvider.getRequestContext();

//Agregar las propiedades de WSAddressing
reqContext.put(JAXWSAConstants.CLIENT_ADDRESSING_PROPERTIES, addrProps);

//Agregar las propiedades de WSSecurity
reqContext.put(AgesicConstants.SAML1_PROPERTY, assertionResponse);

//Instalar los manejadores de WSAddressing y WSSecurity, y el manejador SAML
List<Handler> handlerChain = new ArrayList<Handler>();
handlerChain.add(new WSAddressingClientHandler());
handlerChain.add(new WSSecurityHandlerServer());
handlerChain.add(new SAMLHandler());
//Aquí habría que instalar la lista de maneadores, pero como será necesario
//agregar un manejar más, se deja para más adelante
//bindingProvider.getBinding().setHandlerChain(customHandlerChain);
```

5.3.5 - Instalar el manejador de Publish And Suscribe para recepción

Hasta aquí, el código implementado no es diferente del requerido para invocar cualquier otro servicio expuesto a través de la PGE. Lo que sigue es específico de la recepción de novedades.

A continuación se debe instalar un manejador capaz que colocar en los encabezados del mensaje SOAP los datos del suscriptor y el tópico; este manejador es una instancia de la clase *uy.gub.agesic.novedades.PullHeadersClientHandler*, y se instala de la misma manera que se instalaron los manejadores en el paso previo. Los datos del suscriptor y del tópico se configuran durante la invocación del servicio.

```
PullHeadersClientHandler pullHandler = new PullHeadersClientHandler();
handlerChain.add(pullHandler);

//Ahora si instalar la lista de manejadores
bindingProvider.getBinding().setHandlerChain(handlerChain);
```

5.3.6 - Consumir el Servicio

Finalmente, después de haber realizado toda la configuración necesaria, se puede proceder a consumir el servicio. Al hacerlo, se debe incluir como parámetro de la invocación los datos correspondientes al suscriptor (identificar quién es el que pretende recibir la novedad) y el tópico (desde dónde desea recibir la novedad); además, si ya se recibió una novedad previamente, se debe incluir también el identificador de la misma a modo de ACK, a los efectos de confirmarle a la PGE que dicha novedad se ha recibido correctamente (en caso contrario, la PGE continuará enviando la misma novedad una y otra vez; si es la primera vez que se obtiene una novedad, no hay identificador para pasar, y por tanto se debe pasar null). Luego de consumir el servicio, si la respuesta es nula, es porque no hay novedades que no se hayan recibido previamente, mientras que si no es nula, se ha recibido una instancia de la clase NoticiaType.

```
String suscriptor = "Testing";
String topico = "NoticiasAgesic";
String notificacionId = ... //El identificador de la última novedad recibida
                            //anteriormente, o null si es la primera vez

PullNotificationRequest req = new PullNotificationRequest();
req.setSubscriber(suscriptor);
req.setTopic(topico);
req.setAckNotificationId(notificationId);
NoticiaType noticia = port.recibirNovedad(req);
if (noticia == null){
  System.out.println("No hay nuevas noticias");
}else{
  notificacionId = handler.getNotificationId();
  System.out.println("Noticia recibida OK: " + notificacionId);
  PersonaType persona = noticia.getPersonas().getPersona().get(0);
  System.out.println("Documento: " + persona.getCi());
  System.out.println("Nombre: " + persona.getNombre());
  System.out.println("Apellido: " + persona.getApellido());
}
```

Para ejecutar el cliente implementado hacer clic derecho sobre la clase *ClientSample* y seleccionar la opción *Run as → Java Application*. La consola debería mostrar un mensaje similar a alguno de los presentados, según existan novedades pendientes o no:

```
<<Si no hay alguna noticia pendiente>>
      No hay nuevas noticias


   <<Si hay una noticia pendiente>>
      Noticia recibida OK: XXX
      Documento: DocumentoPersona
      Nombre: NombrePersona
      Apellido: ApellidoPersona
```

En el Apéndice 2 – Recibir Novedad: código fuente completo se puede ver el código fuente completo.

6 - Apéndices

6.1 - Apéndice 1 – Publicar Novedad: código fuente completo

```
package uy.gub.agesic.test.novedades;

import uy.gub.agesic.ps.samples.novedad.enviarnovedad.*;
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;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
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.extensions.addressing.AttributedURIImpl;
import org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler;
import org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer;
import uy.gub.agesic.AgesicConstants;
import uy.gub.agesic.jbossws.SAMLHandler;
import uy.gub.agesic.novedades.PublishSubscribeHeadersHandler;

public class ClientSample {
  public static void main(String[] args) {
    String userName = "user";
    String role = "ou=capacitacion, o=agesic";
    String service =
      "http://test_servicios.pge.red.uy/agesic/servicioPublicacionNovedadesTest";
    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/SecurityTokenServiceProtected";

    //Alias que identifica al certificado que se debe enviar a la PGE dentro del keystore
    String alias =
      "0f026f823ca3597ced3953188b1628de_be45dff3-4f56-4728-8332-77080b0c1c08";
    String keyStoreFilePath =
      "C:\\materiales\\ClientePubSub\\keystores\\agesic_v3.0.keystore";
    String keyStorePwd = "agesic";
    String trustStoreFilePath =
      "C:\\materiales\\ClientePubSub\\keystores\\agesic_v2.0.truststore";
    String trustStorePwd="agesic";

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

```
keyStore.setStorePwd(keyStorePwd);

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

System.setProperty("javax.net.ssl.keyStore", keyStoreFilePath);
System.setProperty("javax.net.ssl.keyStorePassword",keyStorePwd);
System.setProperty("javax.net.ssl.trustStore",trustStoreFilePath);
System.setProperty("javax.net.ssl.trustStorePassword",trustStorePwd);

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,keyStore,trustStore,stsUrl);
  String stringRepresentation= assertionResponse.toString();
} catch (Exception e) {
  e.printStackTrace();
  System.exit(1);
}

NuevaNovedadService_Service service1 = new NuevaNovedadService_Service();
NuevaNovedadService port1 = service1.getNuevaNovedadPort();

//Propiedades para WS-Addressing
AddressingBuilder addrBuilder = SOAPAddressingBuilder.getAddressingBuilder();
SOAPAddressingProperties addrProps =
  (SOAPAddressingProperties)addrBuilder.newAddressingProperties();
String actionStr = "nuevaNovedad";
AttributedURI soapTo = new AttributedURIImpl(service);
AttributedURI soapAction = new AttributedURIImpl(actionStr);
addrProps.setTo(soapTo);
addrProps.setAction(soapAction);

BindingProvider bindingProvider = (BindingProvider)port1;
Map<String, Object> reqContext = bindingProvider.getRequestContext();

reqContext.put(JAXWSAConstants.CLIENT_ADDRESSING_PROPERTIES, addrProps);

//Propiedades WS-Security (SAML Token de PGE)
reqContext.put(AgesicConstants.SAML1_PROPERTY, assertionResponse);

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

String productor = "AGESIC";
String topico = "NoticiasAgesic";

PublishSubscribeHeadersHandler handler =
   new PublishSubscribeHeadersHandler(productor, topico);
handlerChain.add(handler);

bindingProvider.getBinding().setHandlerChain(handlerChain);

PersonaType personaTaller = new PersonaType();
personaTaller.setApellido("ApellidoParticipanteTaller");
personaTaller.setCi("CIParticipanteTaller");
personaTaller.setNombre("NombreParticipanteTaller");

NoticiaType noticia = new NoticiaType();
```

```
    noticia.setPersonas(new NoticiaType.Personas());
    noticia.getPersonas().getPersona().add(personaTaller);
    port1.nuevaNovedad(noticia);

    System.out.println("Novedad Enviada! ID Notificacion:" +
      handler.getNotificationId());
  }
}
```

6.2 - Apéndice 2 – Recibir Novedad: código fuente completo

```
package uy.gub.agesic.test.novedades;

import uy.gub.agesic.ps.samples.novedad.recibirnovedad.*;
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;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
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.extensions.addressing.AttributedURIImpl;
import org.jboss.ws.extensions.addressing.jaxws.WSAddressingClientHandler;
import org.jboss.ws.extensions.security.jaxws.WSSecurityHandlerServer;
import uy.gub.agesic.AgesicConstants;
import uy.gub.agesic.jbossws.SAMLHandler;
import uy.gub.agesic.novedades.PullHeadersClientHandler;;

public class ClientSample {
  public static void main(String[] args) {
    String userName = "user";
    String role = "ou=TEST_SUBSCRIBE_PULL,O=TEST_PE";
    String service =
      "http://test_servicios.pge.red.uy/agesic/servicioRecepcionNovedadesPullTest";
    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/SecurityTokenServiceProtected";

    //Alias que identifica al certificado que se debe enviar a la PGE dentro del keystore
    String alias =
      "0f026f823ca3597ced3953188b1628de_be45dff3-4f56-4728-8332-77080b0c1c08";
    String keyStoreFilePath =
      "C:\\materiales\\ClientePubSub\\keystores\\agesic_v3.0.keystore";
    String keyStorePwd = "agesic";
    String trustStoreFilePath =
      "C:\\materiales\\ClientePubSub\\keystores\\agesic_v2.0.truststore";
```

```
String trustStorePwd="agesic";

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

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

System.setProperty("javax.net.ssl.keyStore", keyStoreFilePath);
System.setProperty("javax.net.ssl.keyStorePassword",keyStorePwd);
System.setProperty("javax.net.ssl.trustStore",trustStoreFilePath);
System.setProperty("javax.net.ssl.trustStorePassword",trustStorePwd);

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,keyStore,trustStore,stsUrl);
  String stringRepresentation= assertionResponse.toString();
} catch (Exception e) {
  e.printStackTrace();
  System.exit(1);
}

RecibirNovedadService_Service service1 = new RecibirNovedadService_Service();
RecibirNovedadService port1 = service1.getRecibirNovedadPort();

//Propiedades para WS-Addressing
AddressingBuilder addrBuilder = SOAPAddressingBuilder.getAddressingBuilder();
SOAPAddressingProperties addrProps =
  (SOAPAddressingProperties)addrBuilder.newAddressingProperties();
String actionStr = "recibirNovedad";
AttributedURI soapTo = new AttributedURIImpl(service);
AttributedURI soapAction = new AttributedURIImpl(actionStr);
addrProps.setTo(soapTo);
addrProps.setAction(soapAction);

BindingProvider bindingProvider = (BindingProvider)port1;
Map<String, Object> reqContext = bindingProvider.getRequestContext();

reqContext.put(JAXWSAConstants.CLIENT_ADDRESSING_PROPERTIES, addrProps);

//Propiedades WS-Security (SAML Token de PGE)
reqContext.put(AgesicConstants.SAML1_PROPERTY, assertionResponse);

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

PullHeadersClientHandler pullhandler = new PullHeadersClientHandler();
customHandlerChain.add(pullhandler);

bindingProvider.getBinding().setHandlerChain(customHandlerChain);

String subscriptor = "Testing";
String topico = "NoticiasAgesic";

PullNotificationRequest req = new PullNotificationRequest();
req.setSubscriber(subscriptor);
```

```
    req.setTopic(topico);

    NoticiaType noticia = port1.recibirNovedad(req);
    if (noticia == null){
      System.out.println("No hay nuevas noticias");
    }else{
      System.out.println("Noticia recibida OK: " + pullhandler.getNotificationId());
      PersonaType personaTaller = noticia.getPersonas().getPersona().get(0);
      System.out.println("Documento: " + personaTaller.getCi());
      System.out.println("Nombre: " + personaTaller.getNombre());
      System.out.println("Apellido: " + personaTaller.getApellido());
    }

    String notificationId = pullhandler.getNotificationId();
    PullNotificationRequest segundo_req = new PullNotificationRequest();
    segundo_req.setSubscriber(subscriptor);
    segundo_req.setTopic(topico);
    segundo_req.setAckNotificationId(notificationId);
    NoticiaType segunda_noticia = port1.recibirNovedad(segundo_req);
    if (segunda_noticia == null){
      System.out.println("No hay nuevas noticias");
    }else{
      System.out.println("Noticia recibida OK: " + pullhandler.getNotificationId());
      PersonaType personaTaller = segunda_noticia.getPersonas().getPersona().get(0);
      System.out.println("Documento: " + personaTaller.getCi());
      System.out.println("Nombre: " + personaTaller.getNombre());
      System.out.println("Apellido: " + personaTaller.getApellido());
    }
  }
}
```

6.3 - Apéndice 3 – 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 por el organismo. El cabezal “action” debe ser obtenido a partir del documento WSDL que describe el servicio, como se describe 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 nombre de la operación, es decir, el mismo valor del atributo “name” del tag “operation”.
    - Si el valor del atributo “soapAction” no es vacío, se debe especificar exactamente dicho valor, respetando mayúsculas y minúsculas.

7 - 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