

## ***Modulo conexión Cliente WS – DGI***

El desarrollo fue echo con eclipse (eclipse-jee-kepler) utilizando herramientas Apache.  
De la misma forma puede hacerse con otro lenguaje que acepte estas librerías porque la conexión es más bien un tema de configuración.

### **Librerías para conexión:**

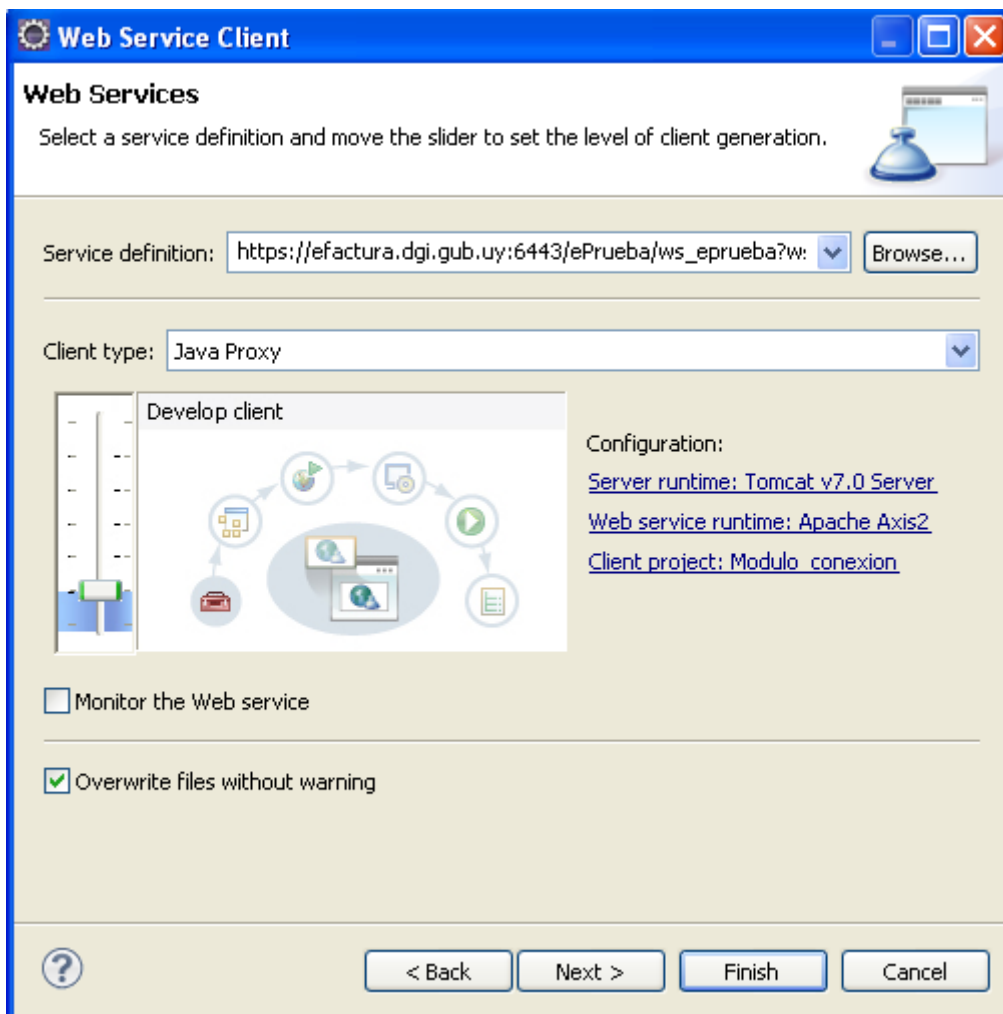
- Axis2 (axis2-1.6.2)
- Rampart (rampart-1.6.2)

### **Librerías para seguimiento (Anexo):**

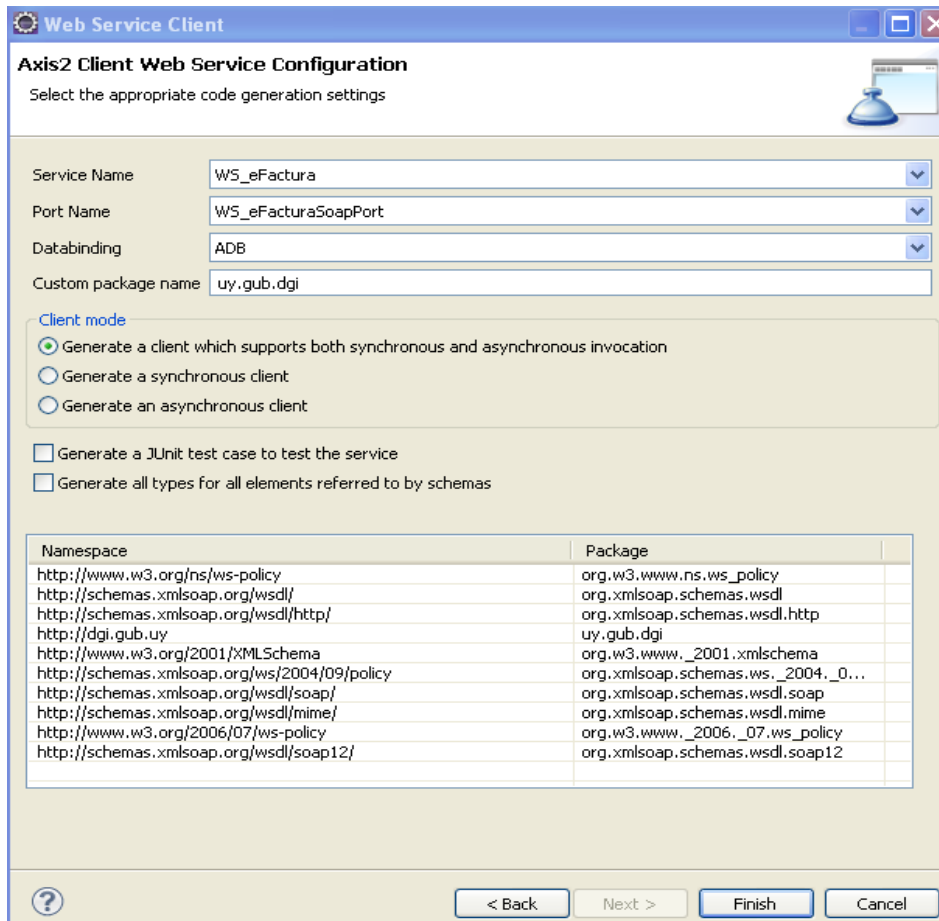
- stunnel-5.03-installer (intermediario para visualizar mensajes Soap)
- tcp/ip monitor (intermediario para visualizar mensajes Soap)

Los pasos seguidos son:

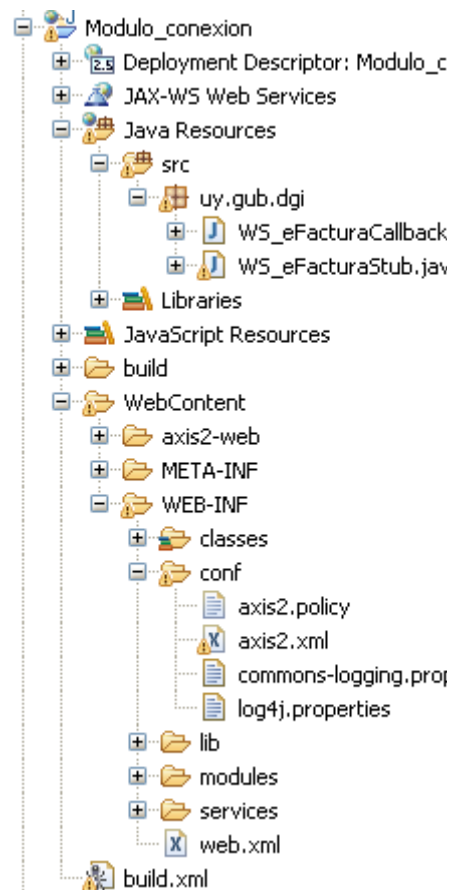
- Armo un nuevo Cliente de Web Service



- Le doy siguiente y no hago nada.....



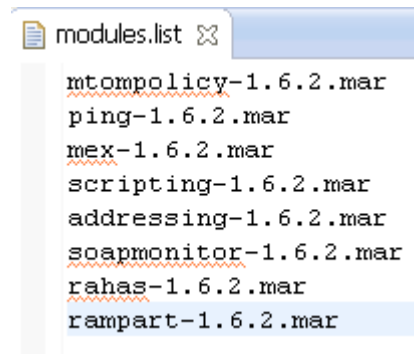
- Finalizo y automaticamente quedan armadas estas carpetas y archivos:



Luego de esto debo configurar el Rampart:

Dentro de este directorio (que se ve en la figura de arriba ) **WebContent/WEB-INF/lib** pego los archivos de librería que se encuentran en **Rampart-1.6.2/lib** (que están dentro de lo que baje de Rampart)

Y dentro del directorio **WebContent/WEB-INF/modules** pego los modulos que están dentro de la carpeta **rampart-1.6.2/modules** (que son rahas-1.6.2 y rampart-1.6.2). Luego de esto agregarlos al listado Modules.list que esta en **WebContent/WEB-INF/modules**



```
modules.list
mtompolicy-1.6.2.mar
ping-1.6.2.mar
mex-1.6.2.mar
scripting-1.6.2.mar
addressing-1.6.2.mar
soapmonitor-1.6.2.mar
rahas-1.6.2.mar
rampart-1.6.2.mar
```

Por algún error del Eclipse dos archivos de la librería de Axis2 no los trae y los tengo que incluir en **WebContent/WEB-INF/lib** pegandolos desde lo que baje de Axis2 dentro de la carpeta **axis2-1.6.2/lib** , los archivos son **jaxen-1.1.1.jar** y **mex-1.6.2-impl.jar**

**Más adelante, en el Anexo muestro como configurar el visor de los mensajes SOAP, que lo armo Maurizio, y que es vital para este desarrollo, ya que sino trabajamos a ciegas**

Una vez se tiene todo cargado se cambia el archivo de configuración **Axis2.xml** que viene por defecto por el que adjunto con el pdf ( se ve en figura de arriba )

Luego de pegado el archivo Axis2.xml hay que abrirlo y setear con sus datos temas de seguridad

De aca en adelante pongo entre [ ] (no van los parentesis) lo que hay que completar:

```
<parameter name="OutflowSecurity">
  <action>
    <items>Signature</items>
    <user>[Usuario del Certificado]</user>
    <signaturePropFile>sec.properties</signaturePropFile>
    <passwordCallbackClass>uy.conexion.Psw_cb</passwordCallbackClass>
    <signatureKeyIdentifier>DirectReference</signatureKeyIdentifier>
  </action>
</parameter>

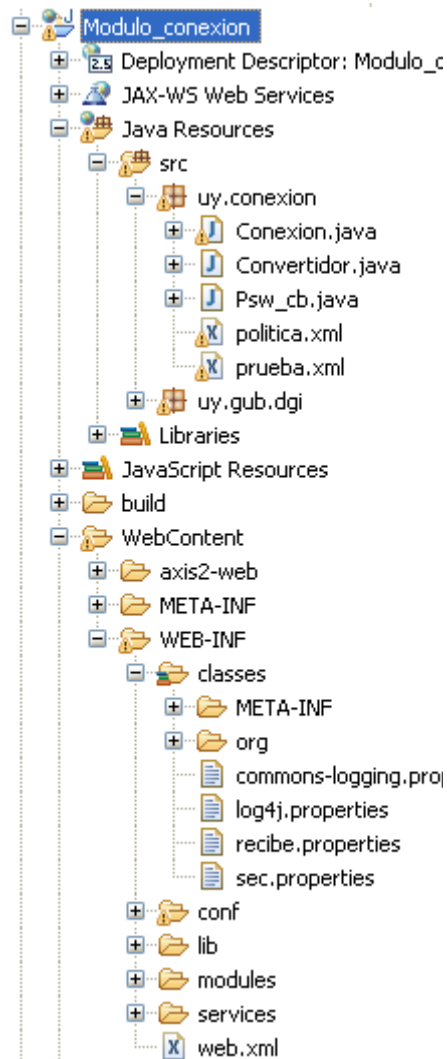
<parameter name="InflowSecurity">
  <action>
    <items>Signs</items>
    <signaturePropFile>recibe.properties</signaturePropFile>
  </action>
</parameter>
```

**sec.properties** - Crypto properties para enviar (va en carpeta **WebContent/WEB-INF/classes** )

**recibe.properties** - Crypto properties para recibir (va en carpeta **WebContent/WEB-INF/classes** )

**uy.conexion.Psw\_cb** - Clase que viene más abajo

Al Proyecto le agrego los archivos en el paquete creado “uy.conexion” y las properties que menciono arriba :



- **Conexion.java** es la que realizara la conexión, paso codigo en detalle a continuación
- **Convertidor.java** convierte para pasar sobre de ejemplo, paso codigo en detalle a continuación
- **Psw\_cb.java** es la que maneja la contraseña, paso código también
- **politica.xml** es la política que le paso al Rampart para que funcione, De aqui toma la configuración para hablar con el certificado y el Security Binding (es lo que esta entre etiquetas `AsymmetricBinding` - maneja los token )
- **prueba.xml** es el sobre de ejemplo para pasar a DGI con los comprobantes

- La clase **WS\_eFacturaStub.java** que esta en el paquete `uy.gub.dgi` que aparece arriba (Maneja toda la conexión y es armada automaticamente con la definicion del servicio que hicimos en el primer paso)

- **sec.properties** – archivo que tienen que llenar con sus datos (Hay que ubicarlo en **WEB-INF/classes** como se ve arriba). El contenido es:

```
org.apache.ws.security.crypto.provider=
org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.keystore.type=JKS
org.apache.ws.security.crypto.merlin.keystore.password=[contraseña]
org.apache.ws.security.crypto.merlin.keystore.alias=[alias certif nuestro]
org.apache.ws.security.crypto.merlin.keystore.file=C:\\...\\archivo.jks
```

- **recibe.properties** – archivo que tienen que llenar con sus datos (Hay que ubicarlo en **WEB-INF/classes** como se ve arriba). El contenido es:

```
org.apache.ws.security.crypto.provider=
org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.keystore.type=JKS
org.apache.ws.security.crypto.merlin.keystore.password=[contraseña]
org.apache.ws.security.crypto.merlin.keystore.alias=[alias certif DGI]
org.apache.ws.security.crypto.merlin.keystore.file=C:\\...\\archivo.jks
```

## **La parte de código:**

### **Conexion.java**

```
package uy.conexion;

import java.io.File;
import java.io.IOException;
import java.rmi.RemoteException;

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

import uy.gub.dgi.WS_eFacturaStub;

public class Conexion {

    public static void main(String[] args) throws
    javax.xml.stream.XMLStreamException, IOException {

        // Configuramos el contexto
        final ConfigurationContext ctx = ConfigurationContextFactory.
            createConfigurationContextFromFileSystem(
                "WebContent/WEB-INF", // Donde están Axis2 y Rampart
                "WebContent/WEB-INF/conf/axis2.xml"); // Ubicacion exacta del
            archivo axis2.xml que maneja la conexión

        final WS_eFacturaStub wse_conexion = new WS_eFacturaStub(ctx);
        //Genero la conexion con la clase de la DGI y el contexto de arriba
```

```

// Aca armo para cargar la politica falsa para que funcione Rampart
final Options options = wse_conexion._getServiceClient().getOptions();

// Cargamos el archivo de la politica Rampart
final StAXOMBuilder builder =
    new StAXOMBuilder("src/uy/conexion" + File.separator
        + "politica.xml");
final Policy policy =
    PolicyEngine.getPolicy(builder.getDocumentElement());
    options.setProperty(RampartMessageData.KEY_RAMPART_POLICY,
        policy);

// Aca armo el envío seteando la clase WS_eFacturaStub
uy.gub.dgi.WS_eFacturaStub.WS_eFacturaEFACRECEPCIONSOBRE sobre = new
uy.gub.dgi.WS_eFacturaStub.WS_eFacturaEFACRECEPCIONSOBRE();
WS_eFacturaStub.Data xml_sobre= new WS_eFacturaStub.Data();

// Aca convierto para adjuntar el xml
Convertidor convertidor = new Convertidor();
xml_sobre.setXmlData(convertidor.convertToXMLData(
    "src/uy/conexion/prueba.xml"));

    sobre.setDatain(xml_sobre);

// Aca invoco al Web Service
    try {

uy.gub.dgi.WS_eFacturaStub.WS_eFacturaEFACRECEPCIONSOBREResponse
respSobre= wse_conexion.eFACRECEPCIONSOBRE(sobre);
        System.out.println(respSobre.getDataout().getXmlData());
    } catch (RemoteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    }
}

```

## Convertidor.java

```

package uy.conexion;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class Convertidor {

    public Convertidor() { }

    public String convertToXMLData(String fileName) throws
        FileNotFoundException, IOException {

        BufferedReader br;
        br = new BufferedReader(new FileReader(fileName));

        int c;
        StringBuilder response= new StringBuilder();

        while ((c = br.read()) != -1) {
            response.append( (char)c );
        }
    }
}

```

```

    }
    String xmlData = response.toString();
    br.close();

    return xmlData;
}
}

```

### Psw\_cb.java

```

package uy.conexion;

import java.io.IOException;

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

import org.apache.ws.security.WSPasswordCallback;

public class Psw_cb implements CallbackHandler {

    public void handle(Callback[] callbacks) throws IOException,
        UnsupportedCallbackException {

        // Con esta Clase manejo el acceso al certificado (Hay que mejorar
        // el tema seguridad con Digest, etc)

        WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[0];

        String id = pwcb.getIdentifier();
        int usage = pwcb.getUsage();

        if (usage == WSPasswordCallback.SIGNATURE || usage ==
            WSPasswordCallback.DECRYPT ) {

            if ("[Usuario del Certificado]".equals(id) ) {
                pwcb.setPassword("[Contraseña del Certificado]");
            }
        }
    }
}

```

Con estas 3 clases y los archivos mencionados queda armada la conexión y el web service acepta el sobre enviado....

```

<ACKSobre xmlns="http://cfe.dgi.gub.uy" version="1.0">
...
<CantidadCFE>50</CantidadCFE>
<Tmst>2014-09-24T11:36:21-03:00</Tmst></Caratula>
<Detalle><Estado>AS</Estado>
...

```

## ANEXO

### Seguimiento mensajes SOAP

Lo último para configurar es la visualización de los mensajes Soap. Sin esta herramienta estaríamos trabajando a ciegas.

Basicamente con el TCP/IP Monitor nos conectamos al Stunnel que se conecta a la DGI con lo cual utilizando este elemento en el medio podemos ver los mensajes.

TCP Monitor se conecta al puerto 8081 y monitorea el puerto 8080  
Stunnel recibe del 8081 y se conecta a efactura.dgi.gub.uy:6443

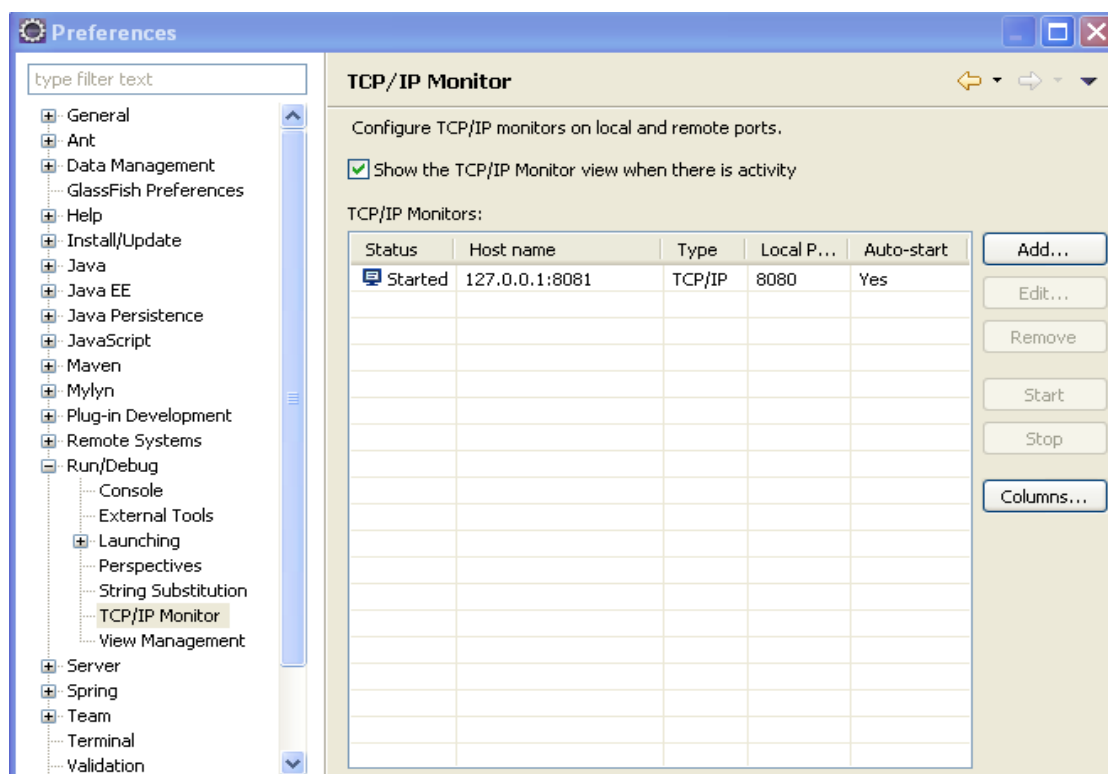
Las configuraciones necesarias son:

#### En el stunnel config

```
client=yes  
verify=0  
debug=7
```

```
[my-https]  
accept = 8081  
connect = efactura.dgi.gub.uy:6443  
TIMEOUTclose = 0
```

#### En el TCP/IP Monitor





Dentro de la clase de la DGI WS\_eFacturaStub.java cambiamos momentaneamente lo siguiente para que el monitoreo funcione:

```
public WS_eFacturaStub(org.apache.axis2.context.ConfigurationContext
configurationContext) throws org.apache.axis2.AxisFault {

    //this(configurationContext,"https://efactura.dgi.gub.uy:6443/ePrueba/ws_e
prueba" );

    this(configurationContext,"http://localhost:8080/ePrueba/ws_eprueba" );

}
```

Con esto queda listo y arroja lo siguiente:

