

Índice

1 Cliente JAVA.....	2
1.1 Wsdl2Java con librerías CXF.....	2
1.1.1 Crear las clases con las librerías cxf.....	2
1.1.2 Ejecutar cliente.....	2
1.2 Wsdl2Java con librerías JAXWS.....	3
1.2.1 Crear las clases con las librerías jaxws:.....	3
1.2.2 Ejecutar cliente:.....	4
1.3 Crear cabeceras de seguridad (CXF o JAX).....	5
1.4 Actualización de PATH y JAVA_HOME.....	8
2 Cliente PERL.....	9
3 Cliente PHP.....	11
4 Cliente Python.....	13
5 Anexo:.....	15
5.1 Crear conexión con SoapUI:.....	15
5.2 SoapUI con firma digital.....	16
5.3 Agregar certificado para conexión SSL desde Java:.....	18
5.3.1 Cómo obtener el certificado con Firefox:.....	18
5.3.2 Cómo obtener el certificado con Internet Explorer:.....	20
5.3.3 Añadir el certificado a la máquina virtual JAVA del cliente.....	21

1 Cliente JAVA

1.1 WsdI2Java con librerías CXF

A continuación se explica como crear y probar un cliente de WsdI con librerías CXF.

Antes de realizar los pasos explicados posteriormente, es recomendable probar si los servicios funcionan correctamente. Para realizar las pruebas hemos utilizado SoapUI, que se puede descargar desde <http://www.soapui.org/>.

Las librerías cxf se pueden descargar desde <http://cxf.apache.org/download.html>

1.1.1 Crear las clases con las librerías cxf.

Comando de ejemplo:

```
$ /home/user/Escritorio/apache-cxf-2.3.2/bin/wsdI2java -exsh true -dex true -impl -p soa.soades.cnpersona  
-client http://soades.upc.edu:8193/Gestioidentitat/ObtenirCNPersona?wsdl
```

Donde:

-p soa.soades.cnpersona1 → Es el nombre del paquete en donde se crearán las clases.

<http://soades.upc.edu:8193/Gestioidentitat/ObtenirCNPersona?wsdl> → Dirección url donde se encuentra el wsdl.

Hay más parámetros disponibles. La lista y descripción de estos se puede encontrar en

<http://cxf.apache.org/docs/wsdI-to-java.html>.














1.1.2 Ejecutar cliente.

Copiar el directorio con las clases creadas dentro del src de nuestro proyecto. Una de las clases creadas tiene el nombre que termina con "..._Client". Esta tiene una función main que podemos ejecutarla en local para comprobar que la conexión se realiza de forma correcta.

1.2 WsdI2Java con librerías JAXWS

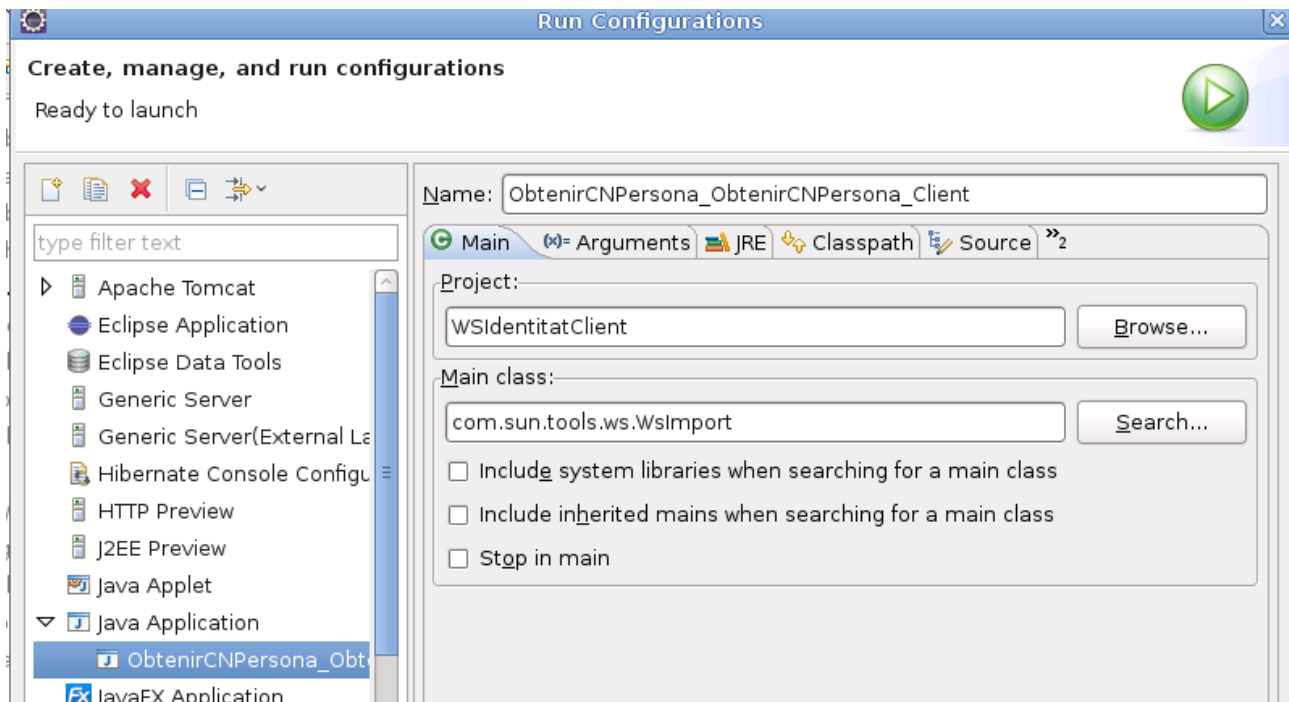
1.2.1 Crear las clases con las librerías jaxws:

Importar las librerías JAXWS al proyecto. Desde el eclipse buscar el `jaxws-tool.jar` → `com.sun.tools.ws` → `WsImport.class`.

- ▷  `jaxrpc.jar` - /home/pablo.gabriel/Escritorio/Pablo/lib
- ▷  `jaxws-api.jar` - /home/pablo.gabriel/Escritorio/Pablo/lib
- ▷  `jaxws-rt.jar` - /home/pablo.gabriel/Escritorio/Pablo/lib
- ▼  `jaxws-tools.jar` - /home/pablo.gabriel/Escritorio/Pablo/lib
 - ▷  `com.sun.istack.ws`
 - ▷  `com.sun.tools.resourcegen`
 - ▼  `com.sun.tools.ws`
 - ▷  `Invoker.class`
 - ▷  `ToolVersion.class`
 - ▷  `WsGen.class`
 - ▷  `WsImport.class`
 -  `version.properties`
 - ▷  `com.sun.tools.ws.ant`

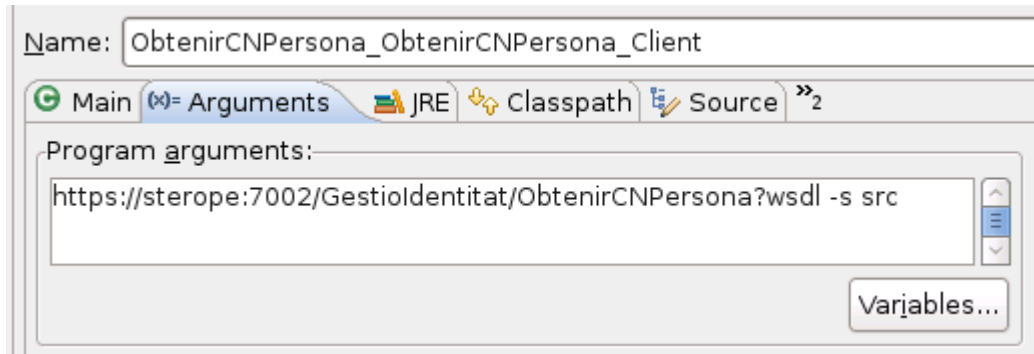
Hacer click en
con el botón
derecho → Run

As → Run Configurations...



Seleccionar en "Project", el proyecto en donde debe generarse el cliente y escribir en "Main Class" "com.sun.tool.ws.WsImport".

En la pestaña de "Arguments" escribir la url del wsdl e indicar con el parámetro "-s" en que carpeta deseamos guardar las clases generadas.



La estructura de las clases generadas es:

- ▼ WSIdentitatRoss
 - ▼ src
 - ▼ com.upcnet.ws.identitat.obtenir_cn_persona
 - ▶ ObjectFactory.java **Clases generadas por librerias jaxws**
 - ▶ ObtenirCNPersona_Type.java **Clases generadas por librerias jaxws**
 - ▶ ObtenirCNPersona.java
 - ▶ ObtenirCNPersonaResponse.java
 - ▶ ObtenirCNPersonaResposta.java
 - ▶ ObtenirCNPersonaService.java
 - ▶ package-info.java
 - ▼ test
 - ▼ com.upcnet.ws.client **Clase de prueba No generada por las librerias**
 - ▶ TestObtenirCNPersona.java
 - ▶ JRE System Library [java-6-sun]
 - ▶ Referenced Libraries
 - ▶ JUnit 4

1.2.2 Ejecutar cliente:

A diferencia de las librerias cxf, esta no genera una clase "Main" para ejecutar el cliente. En el caso del ejemplo se creo la clase "TestObtenirCNPersona", que se puede ejecutar mediante "Junit Test".

```
package com.upcnet.ws.client;
import junit.framework.Assert;
import org.junit.Test;
import com.upcnet.ws.identitat.obtenir_cn_persona.ObtenirCNPersona;
import com.upcnet.ws.identitat.obtenir_cn_persona.ObtenirCNPersonaResposta;
import com.upcnet.ws.identitat.obtenir_cn_persona.ObtenirCNPersonaService;
import com.upcnet.ws.wss.utils.WSSParametresException;
import com.upcnet.ws.wss.utils.WSSUtilities;
```

```

public class TestObtenirCNPersona {

    private String user = "???user???";
    private String password = "???password???";

    @Test
    public void testObtenirCN(){
        ObtenirCNPersonaService servei = new ObtenirCNPersonaService();
        ObtenirCNPersona cn = servei.getObtenirCNPersona();
        try {
            WSSUtilities.afegirUsernameTokenPeticioSOAP(cn, user, password);
        } catch (WSSParametresException e) { Assert.fail(); }
        ObtenirCNPersonaResposta resposta=cn.obtenirCNPersona("??param??");
        System.out.println(resposta.getDocument());
        System.out.println(resposta.getCommonName());
    }
}

```

1.3 Crear cabeceras de seguridad (CXF o JAX).

Si el servicio se encuentra securizado, se deben implementar dos clases para crear las cabeceras que llevarán la información de, por ejemplo, password y usuario. Se debe agregar un par de líneas en la clase de test, justo después de inicializar el “servicio”. Se agrega un “try” para comprobar que los parámetros son correctos.

CXF:

```

try{
    WSSUtilities.afegirUsernameTokenPeticioSOAP(port, "username", "password");
} catch (Exception e) {
    //Tratamiento de la excepción
}

```

JAX:

```

ObtenirCNPersonaService servei = new ObtenirCNPersonaService();
ObtenirCNPersona cn = servei.getObtenirCNPersona();

try{
    WSSUtilities.afegirUsernameTokenPeticioSOAP(cn, username, password);
} catch (Exception e) {
    //Tratamiento de la excepción
}

```

Las clases utilizadas son “WSSUtilities” cuyo código se describe a continuación.

WSSUtilities:

```

package es.upcnet.drac.webservice.client.utils;

import java.util.List;
import java.util.Set;

```

```
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.handler.Handler;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

import org.apache.log4j.Logger;

/**
 * Classe que dona certes utilitats per el desenvolupament de serveis webs amb WSS.
 * @author rodrigo.duran jordi.sala-carrion
 */
public class WSSUtilities {

    private static Logger logger = Logger.getLogger(WSSUtilities.class);

    private static final String SECURITY_NAMESPACE = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd";

    /**
     * Funció que permet afegir a la capçalera de la petició SOAP la informació
     * per l'autenticació de l'usuari que vol executar el servei
     * @param ws - Instància del nostre servei
     * @param username String - Usuari a afegir a la capçalera.
     * @param password String - Contrasenya a afegir a la capçalera.
     * @throws WSSParametresException Excepció bàsica de control de parametres.
     */
    @SuppressWarnings("rawtypes")
    public static void afegirUsernameTokenPeticioSOAP(Object ws, final String username, final String password) throws Exception {
        try {
            if(username == null || password == null){
                throw new Exception("El username i el password han d'estar informats.");
            }
            BindingProvider bp = (BindingProvider) ws;
            bp.getRequestContext().put(BindingProvider.USERNAME_PROPERTY, username);
            bp.getRequestContext().put(BindingProvider.PASSWORD_PROPERTY, password);
            List<Handler> chain = bp.getBinding().getHandlerChain();
            chain.add(createHandler(username, password));
            bp.getBinding().setHandlerChain(chain);
        } catch (SOAPException e) {
            logger.error("Could not configure Username Token Profile authentication ->" + e.getMessage());
        }
    }

    @SuppressWarnings("rawtypes")
    private static SOAPHandler createHandler(final String username,
        final String password) {
        return new javax.xml.ws.handler.soap.SOAPHandler() {
```

```
@Override
public void close(MessageContext context) {
}

@Override
public boolean handleFault(MessageContext context) {
    return false;
}

@Override
public boolean handleMessage(MessageContext context) {
    boolean isOutBound = (Boolean)
context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
    if (!isOutBound) {
        return true;
    }
    try {
        SOAPMessageContext smc = (SOAPMessageContext) context;
        SOAPMessage message = smc.getMessage();
        SOAPHeader header = message.getSOAPHeader();
        if (header == null) {
            SOAPEnvelope envelope = smc.getMessage()
.getSOAPPart().getEnvelope();
            header = envelope.addHeader();
        }
        SOAPElement securityEl = header.addChildElement(new
QName(SEcurity_NAMESPACE, "Security"));
        SOAPElement tokenEl = securityEl.addChildElement(new
QName(SEcurity_NAMESPACE, "UsernameToken"));
        SOAPElement userNameEl = tokenEl.addChildElement(new
QName(SEcurity_NAMESPACE, "Username"));
        userNameEl.addTextNode(username);
        SOAPElement passwordEl = tokenEl.addChildElement(new
QName(SEcurity_NAMESPACE, "Password"));
        passwordEl.addTextNode(password);

        logger.debug(header);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return true;
}

@Override
public Set getHeaders() {
    return null;
}
};
}
}
```

1.4 Actualización de *PATH* y *JAVA_HOME*

Si no se tiene actualizadas estas variables, realizar los siguientes pasos desde consola.

Verificar donde tenemos las librerías java.

```
$ sudo update-alternatives --config java
```

Direccionar *PATH* y *JAVA_HOME* a las librerías (en el caso del ejemplo: `/usr/lib/jvm/java-6-sun/jre/`)

```
$ export PATH=$PATH:/usr/lib/jvm/java-6-sun/jre/  
$ export JAVA_HOME=/usr/lib/jvm/java-6-sun/jre/
```


2 Cliente PERL

Imprescindible tener instalado el paquete SOAP::Lite

Script de ejemplo usando el WS de Personesv1 con el siguiente WSDL:
https://bus-soades.upc.edu/GestiIdentitat/Personesv1?wsdl

Operación: obtenirDadesPersona
Parámetros: commonName("usuari.soa")

```
#!/usr/bin/perl

use strict;
use warnings;
use SOAP::Lite;
use Data::Dumper;
# Para debugar descomenta la linea siguiente
#use LWP::Debug; LWP::Debug::level('+'); SOAP::Lite->import(+trace => 'all');

binmode STDOUT, ":utf8";

# Servicio
my $URI = 'http://soa.identitatdigital.upc.edu/Personesv1'; #targetNameSpace del WSDL
my $PROXY = 'https://bus-soades.upc.edu/GestiIdentitat/Personesv1'; #service name location del WSDL

#Validación BUS
my $HEADER_USERNAME = 'USUARIDELBUS', #usuario del bus upc
my $HEADER_PASSWORD = 'CONTRASENYADELBUS'; #contrasenya del bus upc

# Configuración del cliente
my $client = SOAP::Lite->new (
    uri => $URI,
    proxy => $PROXY
);

#Prefijo del namespace a usar en la request
$client->ns($URI, 'per');

# Configuración cabecera de seguridad WS-Security
my $security = SOAP::Header->name ('wsse:Security')->attr (
    {
        'soapenv:mustUnderstand' => 1,
        'xmlns:wsse'=>'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd'
    }
);

my $userToken = SOAP::Header->name (
    'wsse:UsernameToken' => \SOAP::Header->value (
        SOAP::Header->name ('wsse:Username')->value ($HEADER_USERNAME)->type ("),
        SOAP::Header->name ('wsse:Password')->value ($HEADER_PASSWORD)->type (")->attr (
            {
                'Type' => 'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText'
            }
        ),
    ),
);
```

```
)->attr (  
  {  
    'xmlns:wsu' => 'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-  
1.0.xsd'  
  }  
);
```

Parámetros de la operación del servicio

```
my $QUERY_CN = SOAP::Data->name ('commonName') -> type('string')->value ('usuari.soa');
```

Llamada a la operación del servicio

```
my $result = $client->obtenerDadesPersona($security->value (\$userToken),$QUERY_CN);
```

Resultados

```
unless ($result->fault) {  
  print Dumper($result->valueof('//return'));  
}  
else {  
  print (join (' ', $result->faultcode, $result->faultstring));  
}
```

3 Cliente PHP

Gracias a Jordi Solé Esteve (EPSEB/UPC) por proporcionarnos este código.

Fuentes necesarias:

WSSoapClient.php → Cliente genérico para cualquier WS

guiaDocentPublica.php → Cliente específico para la guiaDocentPublica

soap-wsse.php → Librería de google que deberéis descargar de: <http://code.google.com/p/wse-php/source/browse/soap-wsse.php>

Substituir en guiaDocentPublica.php los parámetros en negrita por sus valores.

guiaDocentPublica.php

```
<?php
```

```
require_once('WSSoapClient.php');
```

```
/* Configuració de SOA */
```

```
$uri = "https://bus-soa.upc.edu/Prisma/GuiaDocentPublicav2?wsdl"; // Adreça del bus SOA
```

```
$usuari = 'USUARIDELBUS'; // Nom d'usuari per validar-se al bus SOA
```

```
$password = 'CONTRASENYADELBUS'; // Contrasenya per validar-se al bus SOA
```

```
/* Valors per defecte */
```

```
$curs = ""; $grup = null; $idioma = 'CA'; $codi = '230451';
```

```
$param = array( 'codiUpc' => $codi, 'curs' => $curs, 'grup' => $grup, 'idioma' => $idioma );
```

```
/* Crida al bus SOA */
```

```
$client = new WSSoapClient($uri, $usuari, $password);
```

```
try {
```

```
    $result = $client->obtenirPDF($param);
```

```
    if ($result->obtenirPDFReturn->error <0) {
```

```
        print_r($result);
```

```
    } else {
```

```
        $content = $result->obtenirPDFReturn->PDF;
```

```
        print_r($content);
```

```
    }
```

```
} catch (SoapFault $exception) {
```

```
    print_r($exception);
```

```
}
```

```
?>
```

WSSoapClient.php

```
require_once('soap-wsse.php');

class WSSoapClient extends SoapClient {

    private $username;
    private $password;

    public function __construct($wsdl, $username, $password, $options = array()) {
        $url = parse_url($wsdl);
        if (isset($url['port'])) {
            $this->_port = $url['port'];
        }
        $this->username = $username;
        $this->password = $password;
        return parent::__construct($wsdl, $options);
    }

    function __doRequest($request, $location, $saction, $version) {
        $doc = new DOMDocument('1.0');
        $doc->loadXML($request);
        $objWSSE = new WSSESoap($doc);
        $objWSSE->addUserToken($this->username, $this->password);

        $parts = parse_url($location);
        if (isset($this->_port)) {
            $parts['port'] = $this->_port;
        }
        $location = $this->buildLocation($parts);

        $this->__last_request = $objWSSE->saveXML();

        return parent::__doRequest($objWSSE->saveXML(), $location, $saction, $version);
    }

    public function buildLocation($parts = array()) {
        $location = "";

        if (isset($parts['scheme'])) {
            $location .= $parts['scheme'].'://';
        }
        if (isset($parts['user']) || isset($parts['pass'])) {
            $location .= $parts['user'].'.'.$parts['pass'].'@';
        }
        $location .= $parts['host'];
        if (isset($parts['port'])) {
            $location .= ':'.$parts['port'];
        }
        $location .= $parts['path'];
        if (isset($parts['query'])) {
            $location .= '?'.$parts['query'];
        }
        }

        return $location;
    }
}
```

4 Cliente Python

Requeriments

Abans de fer l'execució, cal tenir instal·lats els següents paquets de python:

```
suds (pip install suds)
```

```
z3c.suds (pip install z3c.suds)
```

Execució

```
/usr/bin/python guiaDocent.py
```

Contingut del fitxer (guiaDocent.py)

Cal modificar el contingut en negreta segons convingui. L'exemple es connecta al BUS al servei de Guies Docents i genera un PDF segons el codi d'escola i dades utilitzats.

```
# -*- coding: utf-8 -*-  
  
import sys  
  
from suds.wsse import UsernameToken  
from suds.wsse import Security  
from z3c.suds import get_suds_client  
  
# Usuari del Bus SOA  
bussoa_user = 'XXXXXXXX'  
  
# Contrasenya del Bus SOA  
bussoa_password = 'XXXXXXXX'  
  
# Definició del servei GuiaDocentPublica (WSDL)  
wsdl_guiadocentpublica = 'https://bus-soa.upc.edu/Prisma/GuiaDocentPublicav2?wsdl'  
  
def main(argv):  
  
    client = get_suds_client(wsdl_guiadocentpublica)  
  
    security = Security()  
  
    token = UsernameToken(bussoa_user, bussoa_password)  
  
    security.tokens.append(token)
```

```
client.set_options(wsse=security)

PDFfile = client.service.obtenirPDF(codiUpc=14742, curs=2009, grup=1, idioma='ca')

file = open('guia_docent2009-14742-catala.pdf', 'w')

file.write(PDFfile.PDF)

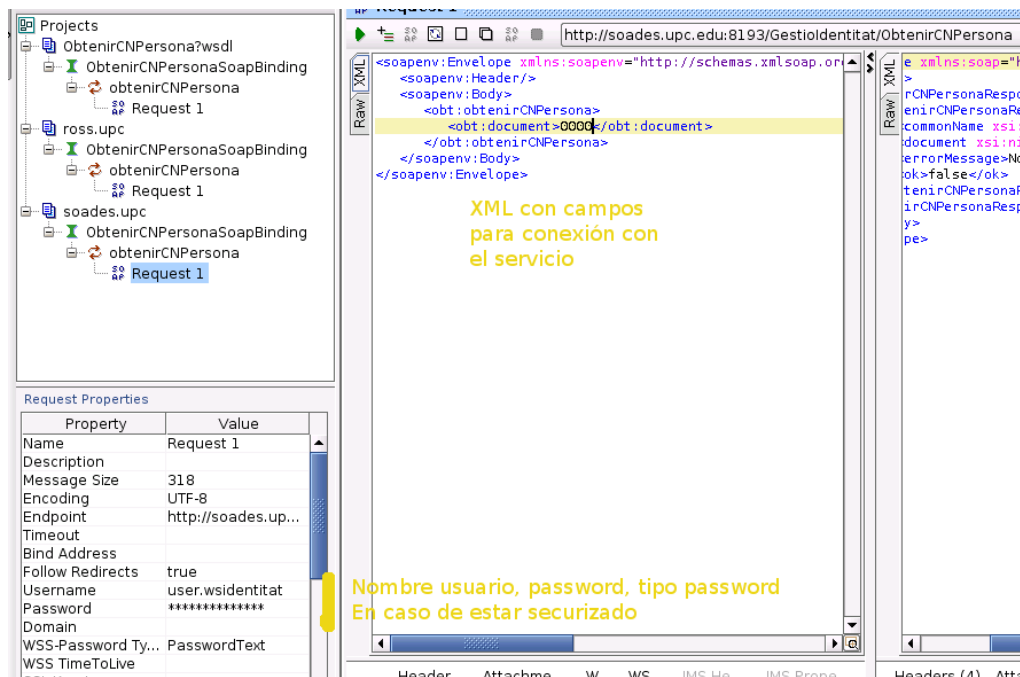
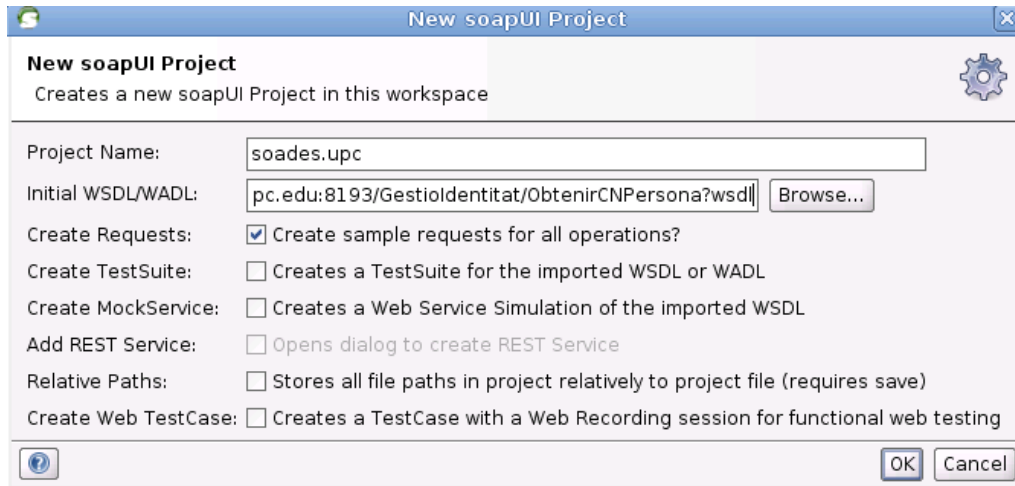
file.close()

if __name__ == "__main__":
    main(sys.argv[1:])
```

5 Anexo:

5.1 Crear conexión con SoapUI:

Con la misma url del wsdl, creamos un nuevo proyecto.

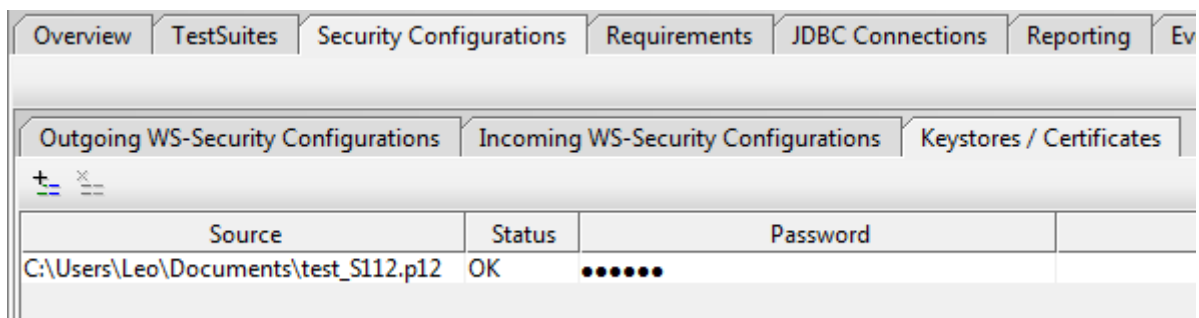


5.2 SoapUI con firma digital

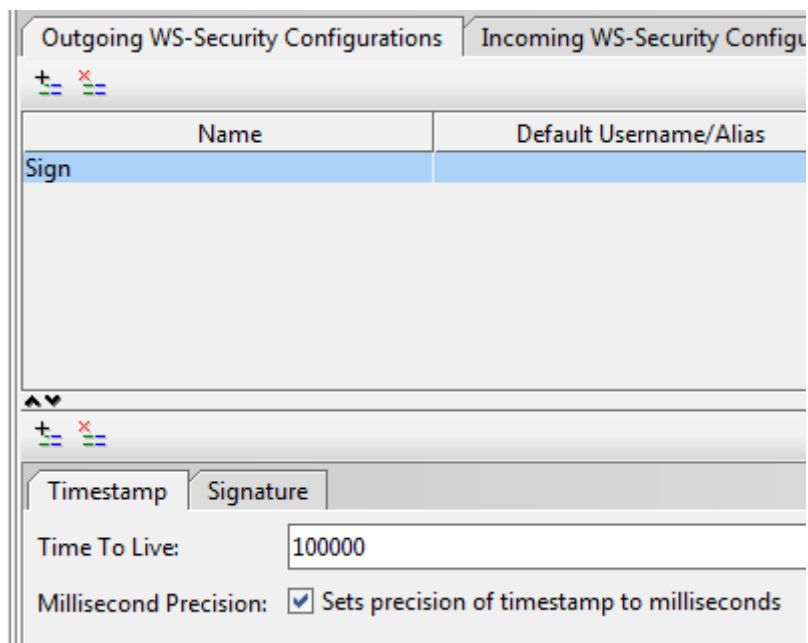
Si quieres configurar SoapUI para que firme la request con un certificado digital hay que seguir los siguientes pasos:

Doble click en el servidor. Pestaña Security Configurations:

1.- Dar de alta el keystore



2.- En la pestaña Outgoing WS-Security Configurations crear una configuración (p.e. Sign) y añadirle el campo Timestamp como se ve en la figura.



3.- Añadir el campo Signature

Keystore: El que habéis dado de alta al principio de todo.

Alias: Certificado a usar para firmar que se encuentra dentro de ese keystore.

Password: Contraseña del certificado

Key identifier type: Binary Security Token

Signature Algorithm: rsa-sha1

Canonicalization: xml-exc-c14n

Parts

Timestamp:

<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd>

Body:

<http://schemas.xmlsoap.org/soap/envelope/>

Token (no namespace)

Name	Default Username/Alias	Default Password
Sign		

Timestamp
Signature

Keystore:

Alias:

Password:

Key Identifier Type:

Signature Algorithm:

Signature Canonicalization:

Use Single Certificate: Use single certificate for signing

Parts:

ID	Name	Namespace	Encode
	Timestamp	http://docs.o...	
	Body	http://schem...	
	Token		

Finalmente, al rellenar la request pulsar el botón “Auth Button” y escoger dentro de “Outgoing WSS” la configuración creada (p.e. Sign).

5.3 Agregar certificado para conexión SSL desde Java:

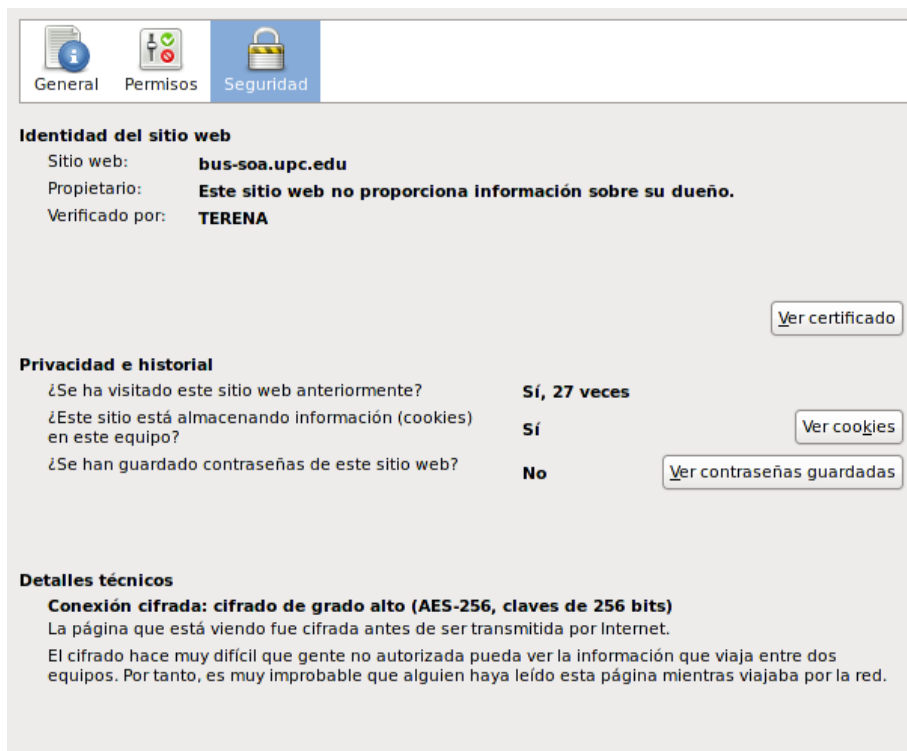
Para conectar con el bus es necesario tener instalado en la máquina virtual de JAVA que usa el cliente el certificado del bus ya que la conexión es SSL.

5.3.1 Cómo obtener el certificado con Firefox:

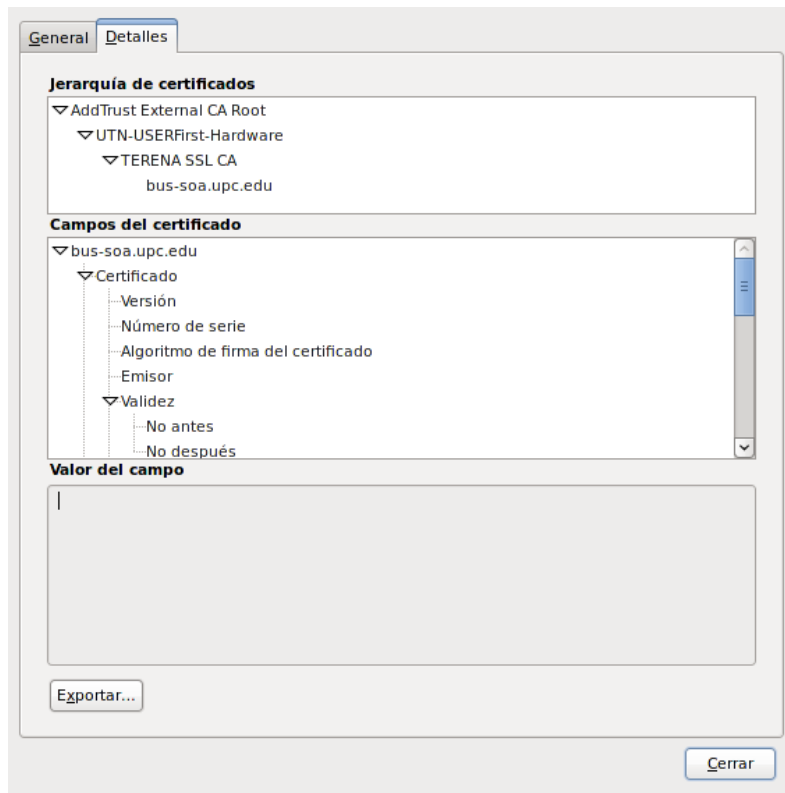
Cargar en firefox la URL del WSDL al que queremos acceder. Pulsar al lado izquierdo de la url.



luego a “Más información”



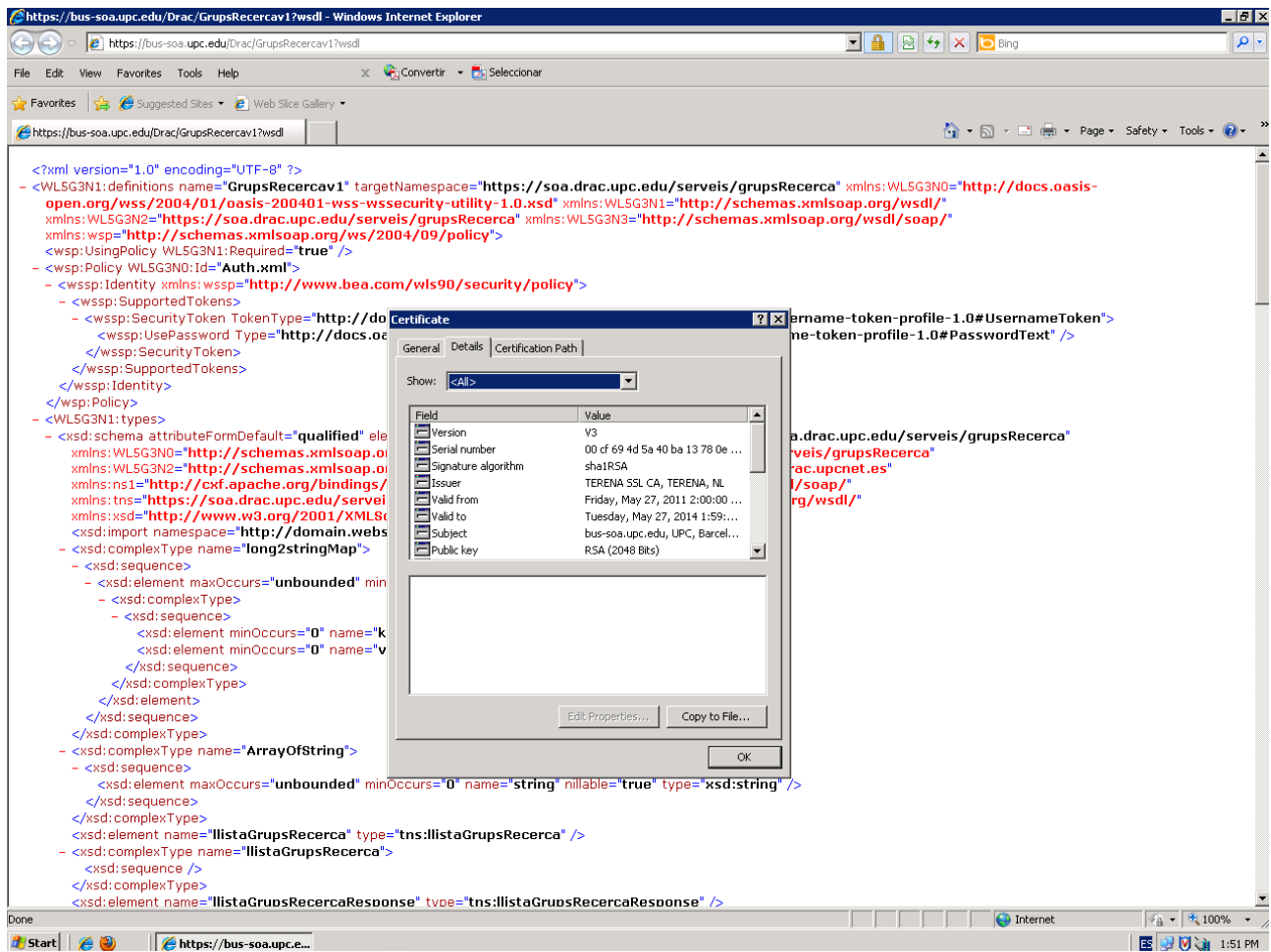
a continuación a “Ver certificado” y pestaña “Detalles”



Y finalmente pulsamos exportar. Con esto guardaremos en disco un fichero que en el caso de ejemplo se llamará bus-soa.upc.edu que contiene el certificado.

5.3.2 Cómo obtener el certificado con Internet Explorer:

Cargamos la URL del WSDL al que queremos acceder. Pulsamos en el candado que podemos encontrar al lado de la url (es posible que en otras versiones esté en otro sitio), después en “ver certificados” y nos aparecerá la siguiente ventana



Vamos a la pestaña detalles y pulsamos “Copy to file” y aceptamos las opciones por defecto que nos diga y lo grabamos en disco.

5.3.3 Añadir el certificado a la máquina virtual JAVA del cliente

Finalmente tendremos que añadirlo al almacén de certificados de la máquina virtual del cliente JAVA que usemos.

```
$ sudo keytool -import -trustcacerts -file bus-soades.upc.edu.pem -keystore  
PATH_JAVA/jre/lib/security/cacerts
```