

---

# **Manual para Desarrolladores de Emisores SCSP J2EE 4.23.0**

**Proyecto:** Librerías SCSP 4.23.0

**Título:** Manual para Desarrolladores de Emisores SCSP J2EE 4.23.0

**Revisión:** 2.17

**Fecha:** Junio de 2020



## Ficha del documento

Fecha	Revisión	Autor	Verificado por
08/04/2014	1.8	Oficina Técnica	

## Documentos relacionados

Fecha	Revisión	Título
23/03/2020	4.15	Requisitos de configuración y despliegue del Portfolio J2EE SCSP v4.23.0.pdf
23/03/2020	2.10	Guía de instalación y despliegue del Portfolio SCSP J2EE-v4.23.0.pdf

## Lista de distribución del documento

Fecha	Nombre
03/04/2014	Administraciones Públicas

## Control de versiones

Fecha	Revisión	Descripción del cambio
03/07/2017	2.5	Actualización a la versión 4.0.0 de las Librerías SCSP
16/10/2017	2.6	Actualización a la versión 4.2.0 de las Librerías SCSP
06/11/2017	2.7	Actualización a la versión 4.3.0 de las Librerías SCSP
30/01/2018	2.8	Actualización a la versión 4.6.0 de las Librerías SCSP
07/05/2018	2.9	Actualización a la versión 4.8.0 de las Librerías SCSP
13/07/2018	2.10	Actualización a la versión 4.11.1 de las Librerías SCSP
09/11/2018	2.11	Actualización a la versión 4.15.0 de las Librerías SCSP
29/03/2019	2.12	Actualización a la versión 4.18.0 de las Librerías SCSP
04/07/2019	2.13	Actualización a la versión 4.20.0 de las Librerías SCSP
20/12/2019	2.14	Actualización a la versión 4.21.0 de las Librerías SCSP

23/03/2020	2.15	Actualización a la versión 4.22.0 de las Librerías SCSP
28/05/2020	2.16	Corrección de erratas
05/06/2020	2.17	Actualización a la versión 4.23.0 de las Librerías SCSP



© 2020 *Ministerio de Asuntos Económicos y Transformación Digital*

Reservados todos los derechos. Quedan rigurosamente prohibidas, sin el permiso escrito de los titulares del copyright, la reproducción o la transmisión total o parcial de esta obra por cualquier procedimiento mecánico o electrónico, incluyendo la reprografía y el tratamiento informático, y la distribución de ejemplares mediante alquiler o préstamos públicos.

This work is protected by copyright. All rights reserved for reproduction or copying of this document or parts thereof. This also applies to its translations. No parts of this work may, in any form whatsoever, (print, photocopy, microfilm or any other procedures), including for training purpose, be reproduced or electronically processed, duplicated or disseminated without the written permission of the copyright owner.

## Contenido

FICHA DEL DOCUMENTO.....	2
DOCUMENTOS RELACIONADOS .....	2
LISTA DE DISTRIBUCIÓN DEL DOCUMENTO .....	2
CONTROL DE VERSIONES.....	2
1 INTRODUCCIÓN.....	7
2 ARQUITECTURA Y DEPENDENCIA DE LAS LIBRERÍAS. ....	7
2.1 Axis .....	7
2.2 Spring .....	7
2.3 Rampart .....	8
3 REQUISITOS TÉCNICOS .....	8
4 INTEGRACIÓN SCSP CON APLICACIONES .....	9
4.1 Configuración en BBDD. ....	9
4.2 Estructura del archivo Recursos_Integracion_Emisor_Librerias_SCSP_4.X.X.rar ...	9
4.3 Librerías necesarias para el desarrollo. ....	10
4.4 Archivos de configuración necesarios para un emisor SCSP. ....	10
4.4.1 Archivo log4j.properties.....	12
4.4.2 Archivo scsp-service.properties.....	13
4.4.3 Archivo axis2.xml.....	14
5 CREACIÓN DEL BACKOFFICE.....	15
6 INTERFACE BACKOFFICE .....	16
6.1 Integración de nuestro BackOffice con las librerías .....	20
7 APÉNDICE .....	22
7.1 Emisor que simula el servicio AEAT103I.....	22
7.1.1 Características .....	22

---

7.1.2	Programación y simulación del emisor AEAT103I .....	24
7.1.3	Como probar la simulación del servicio. ....	27
<b>7.2</b>	<b>Emisor que simula el servicio Q2827003ATGSS006 .....</b>	<b>30</b>
7.2.1	Características .....	30
7.2.2	Programación y simulación del emisor Q2827003ATGSS006 .....	32
7.2.3	Como probar la simulación del servicio. ....	35
<b>7.3</b>	<b>Emisor que simula el servicio Q2827003ATGSS001 .....</b>	<b>37</b>
7.3.1	Funcionamiento Síncrono .....	37
7.3.1.1	Características .....	37
7.3.1.2	Programación y simulación del emisor Q2827003ATGSS001 versión síncrona .....	40
7.3.1.3	Como probar la simulación del servicio. ....	44
7.3.2	Funcionamiento asíncrono .....	46
7.3.2.1	Características .....	46
7.3.2.2	Programación y simulación del emisor Q2827003ATGSS001 versión síncrona. ....	47
7.3.2.3	Como probar la simulación del servicio.....	48
<b>7.4</b>	<b>Emisor con Backoffice de Pruebas .....</b>	<b>51</b>
7.4.1	Funcionamiento Asíncrono .....	51
7.4.1.1	Programación y simulación del emisor que parsea XML versión asíncrona .....	51
7.4.1.2	Como probar la simulación del servicio.....	55

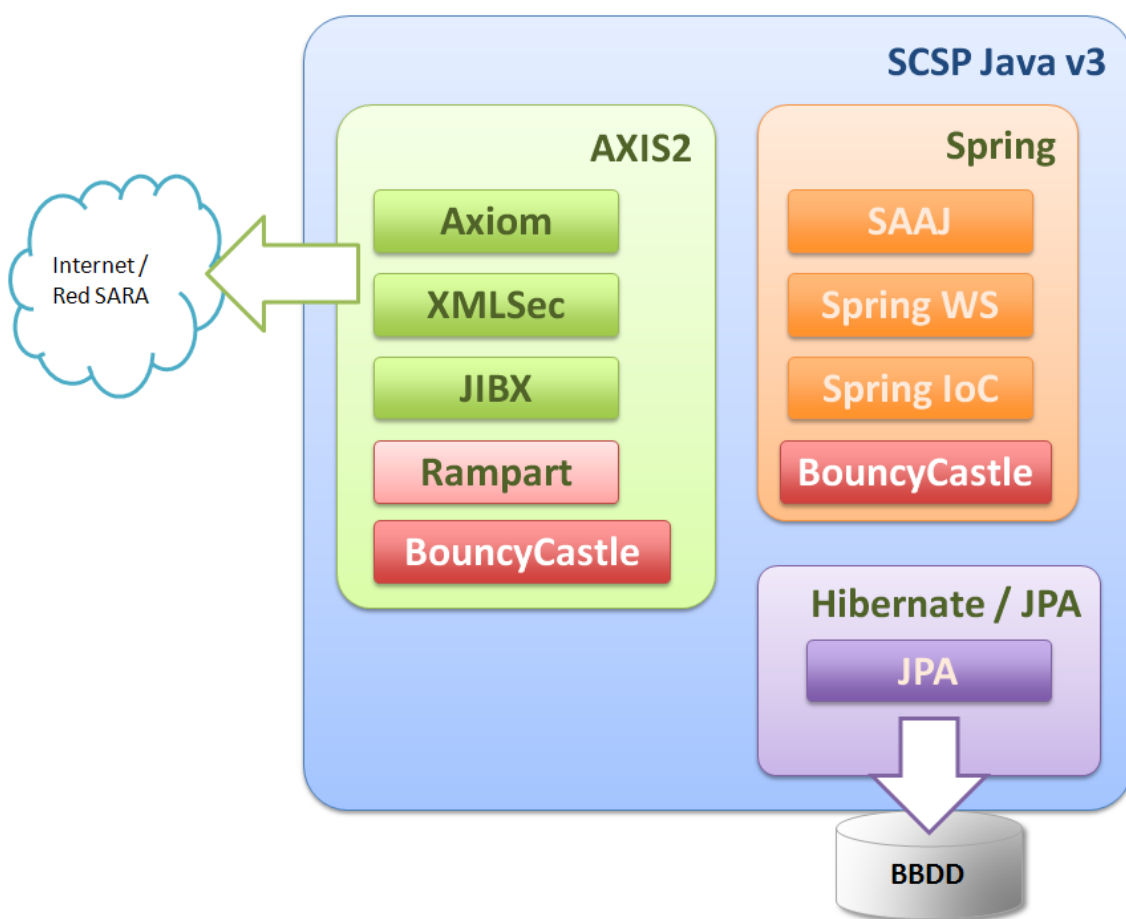
## 1 Introducción

Este documento informa del procedimiento a seguir por los desarrolladores para implementar clases que, desde dentro de sus aplicaciones, consumen los servicios de las librerías SCSP v4.23.0, (desde ahora Librerías SCSP).

Este documento va dirigido a desarrolladores de aplicaciones que deban implementar en sus sistemas peticiones de datos a organismos públicos, y que utilicen para ello las Librerías SCSP.

## 2 Arquitectura y dependencia de las librerías.

Las Librerías SCSP presentan una serie de dependencias con diferentes *thirdpartys* para su correcto funcionamiento. A continuación se muestra un diagrama con los diferentes recursos necesarios



### 2.1 Axis

Las librerías SCSP usan las librerías de Axis en su versión 2.0. Se ha constatado que las versiones de Axis 1.X y 2.0 pueden convivir en un mismo aplicativo sin que por ello existan incompatibilidades.

### 2.2 Spring

Las librerías utilizan una versión de Spring 4.2.0.

Si el aplicativo que se integre con las librerías utiliza una versión anterior o posterior de Spring, deberá revisar la documentación de Spring para conocer el alcance de compatibilidad entre versiones y/o las acciones que se requieran realizar si fuese necesario para actualizar de una versión a otra.

## 2.3 Rampart

Las librerías utilizan la versión 1.6.4 de estas librerías, habiendo sido modificadas las fuentes de Rampart para subsanar ciertas incompatibilidades con las necesidades de la aplicación. El aplicativo que se integre con las librerías SCSP deberá respetar el archivo de Rampart distribuido (***scsp-rampart-core-1.6.4.1.jar***), no siendo sustituido por otro aunque pertenezca a la misma versión 1.6.4.

## 3 Requisitos Técnicos

Para ver las restricciones técnicas y las distintas características arquitectónicas de las librerías SCSP, véase *“Requisitos de configuración y despliegue del Portfolio J2EE SCSP v4.23.0.pdf”*.



## 4 Integración SCSP con aplicaciones

### 4.1 Configuración en BBDD.

A partir de la versión 3.2.0 de las librerías, no existe configuración en ficheros, si no que toda la configuración se encuentra en base de datos, salvo la configuración propia de la conexión contra la base de datos, para ello véase *“Manual de instalación despliegue y actualización del Portfolio SCSP J2EE.pdf” en el apartado 5.1.3 Instalación*.

Puesto que si no se configura previamente la base de datos, los desarrolladores no podrán ejecutar sus pruebas unitarias, para comprobar que los desarrollos realizados son correctos.

### 4.2 Estructura del archivo

#### Recursos\_Integracion\_Emisor\_Librerias\_SCSP\_4.X.X.rar

Este archivo contiene todos los archivos y carpetas necesarias para poder llevar a cabo el desarrollo de un emisor, la estructura del contenido de este fichero es la siguiente:

	Tipo	Descripción
<b>Librerías (*.jar)</b>	Archivos .jar	Librerías necesarias (thirdparty) para el funcionamiento de las librerías.
<b>log4j.properties</b>	Archivo .properties	Archivo de configuración.
<b>scsp-service.properties</b>	Archivo .properties	Archivo de configuración
<b>applicationContext.xml</b>	Archivo .xml	Archivo de configuración
<b>Web.xml</b>	Archivo .xml	Fichero para el despliegue de la aplicación
<b>Weblogic.xml</b>	Archivo .xml	Fichero para el despliegue de la aplicación
<b>conf/axis2.xml</b>	Archivo .xml	En la carpeta conf se encuentra el archivo con la configuración de Axis2
<b>services</b>	Directorio	Directorio con la configuración de los servicios que proporciona un emisor SCSP
<b>modules</b>	Directorio	Carpeta que contiene los archivos .mar necesarios para el funcionamiento de las librerías Axis2

Para conseguir este archivo se deberá descargar desde internet, a través de la url <http://administracionelectronica.gob.es/es/ctt/scsp> en la pestaña de *Área de Descarga* para poder realizar esta acción deberá estar previamente registrado.

### 4.3 Librerías necesarias para el desarrollo.

La lista de librerías que necesitan las Librerías SCSP para funcionar como un emisor se encuentran en el fichero “**Recursos\_Integracion\_Emisor\_Librerias\_SCSP\_4.X.X.rar**”.

Debido a que el emisor proporciona operaciones en forma de servicio web y que son consumidos por los respectivos clientes, el emisor sólo puede ser una aplicación web por lo que las librerías sólo pueden estar en una ubicación:

Tipo de aplicación	Ubicación
Web	WEB-INF/lib

### 4.4 Archivos de configuración necesarios para un emisor SCSP.

Debido a que un Emisor SCSP sólo puede ser una aplicación web, hay una serie de archivos de configuración que, dependiendo de qué tipo de aplicación sea, deberá ubicarse en sitios distintos.

La lista de sitios por archivo es la siguiente:

Fichero	Ap. Web
log4j.properties	WEB-INF/classes
scsp-service.properties	WEB-INF/classes
applicationContext.xml	WEB-INF

**\*\*\* Nota:** El archivo “applicationContext.xml” es un fichero que contiene el contexto de Spring empleado por las librerías. Puede ser modificados para añadir las clases Java Beans necesarias para implementarla lógica de negocio.

Este archivo se encuentra en el archivo de la distribución  
“**Recursos\_Integracion\_Emisor\_Librerias\_SCSP\_4.X.X.rar**”

Por otro lado, las Librerías SCSP usan las librerías de Axis en su versión 2.0. Se ha constatado que las versiones de axis 1.X y 2.0 pueden convivir en un mismo aplicativo, sin que por ello existan incompatibilidades.

El archivo de configuración para la gestión de axis se denomina **axis2.xml** y debe cumplir las siguientes características:

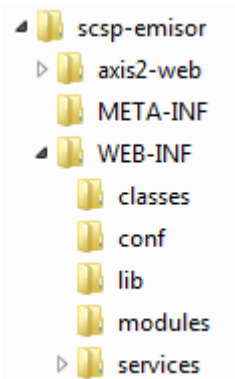
- Debe colocarse en la siguiente ruta:  
WEB-INF/conf/axis2.xml
- Este fichero es estático, por lo que **NO debe modificarse**.

Por otro lado, para axis2 existen una serie de módulos que son los encargados de realizar distintas operaciones sobre los flujos, tanto de entrada o de salida de la comunicación. Se debe validar que existen los siguientes módulos:

- addressing-1.5.mar

- axis2-scripting-1.5.mar
- mex-1.5.mar
- mtompolicy-1.5.mar
- ping-1.5.mar
- scsp-almacenar-fichero-plain.mar
- scsp-almacenar-fichero.mar
- scsp-almacenar-peticion.mar
- scsp-autorizar-organismo.mar
- scsp-global.mar
- scsp-rampart.mar
- scsp-validar-certificado.mar
- scsp-validar-esquema.mar
- scsp-validar-peticion-asincrona.mar
- modules.list
- axis2-jaxws-mar-1.5.mar

La estructura de ficheros que se debe seguir para el fichero de configuración de axis2.xml, y los módulos enumerados en la lista anterior, es la siguiente:



En la carpeta **conf** se encontrará el archivo **axis2.xml**, mientras que en la carpeta modules se encontrarán todos los módulos para realizar la comunicación.

Tanto dicha carpeta como la de **services** deberán situarse bajo **WEB-INF** dentro del contexto de la aplicación web.

#### 4.4.1 Archivo log4j.properties

Se encuentra en la carpeta “**Recursos\_Integracion\_Emisor\_Librerias\_SCSP\_4.23.0.rar**” y se encarga de configurar las propiedades para el uso de los logs dentro de la aplicación, dentro de este archivo no hay propiedades dinámicas, es decir que puedan cambiar, sino que son estáticas, ya que la localización del archivo de logs así como el nivel de la traza, son cogidos de la configuración de base de datos:

NOMBRE	VALOR	TABLA
pathLogs	Path Completo	CORE_PARAMETROS_CONFIGURACION
nivelTraza	DEBUG, ERROR, INFO, WARN o FATAL	CORE_PARAMETROS_CONFIGURACION

El parámetro pathLogs nos indica la ruta de disco donde se quieren dejar los archivos de log. En caso de no existir, dicha carpeta se creará automáticamente. Por otro lado el parámetro niveltraza nos pide una cadena de caracteres con el nivel de traza, en este parámetro sólo se podrá utilizar uno de los siguientes valores, **DE MÁS A MENOS** nivel de traza:

**DEBUG, ERROR, INFO, WARN o FATAL**

Además en la siguiente línea se debe dar de alta el nombre del appender de la siguiente forma:

```
log4j.rootCategory=WARN, CONSOLE, LOGFILE
```

Un ejemplo de configuración de este archivo sería el siguiente:

```
log4j.rootCategory=WARN, CONSOLE, LOGFILE

log4j.logger.es.scsp=DEBUG

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=%-5p - %d{HH:mm:ss} [%c] %m%n

log4j.appender.LOGFILE=org.apache.log4j.RollingFileAppender
log4j.appender.LOGFILE.File=/user/local/tomcat/scsp-ws.log
log4j.appender.LOGFILE.MaxFileSize=1000KB
log4j.appender.LOGFILE.MaxBackupIndex=10
log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
log4j.appender.LOGFILE.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} [%c] %-5p - %m%n
```

## 4.4.2 Archivo scsp-service.properties

Se encuentra en la carpeta “**Recursos\_Integracion\_Emisor\_Librerias\_SCSP\_4.X.X.rar**”, y es el encargado de configurar las propiedades de acceso a la base de datos donde se encuentra la configuración de las Librerías SCSP.

Existen dos posibilidades a la hora de configurar este archivo.

1.- Que tengamos en la misma base de datos del emisor el BackOffice desarrollado

Los campos de este archivo son los siguientes:

- dataSource.driverClassName → nombre de la clase que intermedia entre el código JAVA y la base de datos física.
- dataSource.url → Dirección de la máquina donde se encuentra situado el gestor de la base de datos.
- dataSource.username → Nombre de usuario para acceder a la base de datos.
- dataSource.password → Password para acceder a la base de datos.
- dataSource.validationQuery → Query de validación de BBDD

GESTOR BBDD	QUERY DE VALIDACIÓN
MySQL	select 1
Oracle	select 1 from dual
Sql Server	select 1
PostgreSQL	select 1
MariaDB	select 1

- jdbcName → Nombre JNDI para establecer la conexión por este método. Si está activada se tomará la conexión por JNDI
- hibernate.dialect → Nombre de la clase de HIBERNATE que se encarga de la traducción de las instrucciones de hibernate a SQL para interaccionar con la base de datos. Se puede elegir entre las siguientes:

GESTOR BBDD	DIALECTO HIBERNATE
MySql	org.hibernate.dialect.MySQL5InnoDBDialect
Oracle	org.hibernate.dialect.Oracle10gDialect org.hibernate.dialect.Oracle12cDialect
Sql Server	org.hibernate.dialect.SQLServer2005Dialect org.hibernate.dialect.SQLServer2008Dialect org.hibernate.dialect.SQLServer2012Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQL82Dialect org.hibernate.dialect.PostgreSQL9Dialect

	org.hibernate.dialect.PostgreSQL94Dialect
	org.hibernate.dialect.PostgreSQL95Dialect
<b>MariaDB</b>	org.hibernate.dialect.MySQL5InnoDBDialect

No se debe utilizar el dialect **org.hibernate.dialect.SQLServerDialect** al estar obsoleto y al contener bugs que hacen que la aplicación no funciona correctamente

- hibernate.dbcp.enabled → Activación del pool de conexiones c3p0

Las siguientes propiedades debe existir **NO** deben ser modificadas:

- hibernate.show\_sql → false
- hibernate.hbm2ddl.auto → auto

## 2.- Que tengamos en distinta base de datos del emisor el BackOffice desarrollado

Para este caso, debemos añadir las siguientes propiedades con sus parámetros correctos

- dataSource.driverClassNameBackOffice → nombre de la clase que intermedia entre el código JAVA y la base de datos física.
- dataSource.urlBackOffice → Dirección de la máquina donde se encuentra situado el gestor de la base de datos.
- dataSource.usernameBackOffice → Nombre de usuario para acceder a la base de datos.
- dataSource.passwordBackOffice → Password para acceder a la base de datos.
- jndiNameBO → Nombre JNDI para establecer la conexión por este método. Si está activada se tomará la conexión por JNDI
- hibernate.dialectBackOffice → Nombre de la clase de HIBERNATE que se encarga de la traducción de las instrucciones de hibernate a SQL para interaccionar con la base de datos del BackOffice. Se puede elegir entre las mismas propiedades que para el emisor descritas anteriormente.

### 4.4.3 Archivo axis2.xml

Este archivo se encarga del control de las comunicaciones realizadas entre las librerías SCSP 4.X.X y los distintos organismos a través del Framework axis 2, en él también se indican que manejadores se van a utilizar, y el orden tanto para los flujos de entrada y salida.

Este archivo es estático, no debe ser modificado y se encuentra en la carpeta "Recursos\_Integracion\_Emisor\_Librerias\_SCSP\_4.X.X.rar".

## 5 Creación del BackOffice

Con las Librerías SCSP se distribuye con una aplicación web denominada “**scsp-emisor**”, que contiene los elementos necesarios para desplegar un emisor de servicios SCSP.

Dicho emisor no es más que un proyecto que utiliza Axis2 como motor de servicios web para exponer los tres métodos web utilizados por SCSP:

- Peticiones síncronas
- Peticiones asíncronas
- Solicitud asíncrona de respuestas

El proyecto que se distribuye con las librerías, por sí mismo carece de funcionalidad al no tener ningún BackOffice que realice la lógica de negocio del servicio. Por lo tanto, tenemos que esta aplicación web no es más que un entorno que permitirá a los integradores realizar sus propios desarrollos de emisores.

No obstante, las librerías contienen una serie de implementaciones de demostración que pueden ser utilizados en entornos de pruebas.

## 6 Interface BackOffice

Implementar un emisor utilizando las librerías es tan sencillo como realizar una implementación de la clase BackOffice definida dentro del jar “scsp-core”. La definición de esta clase es la siguiente:

```
package es.scsp.common.backoffice;

public abstract class BackOffice {

    abstract Respuesta NotificarSincrono(Peticion obj) throws ScspException;

    abstract ConfirmacionPeticion NotificarAsincrono(Peticion obj) throws ScspException;

    abstract Respuesta SolicitudRespuesta(SolicitudRespuesta obj) throws ScspException;

}
```

Tal y como podemos ver, no hace más que definir una interface para cada uno de los métodos utilizados por SCSP que se expondrán a través de los servicios web.

Los objetos del modelo SCSP (**Peticion**, **Respuesta**, **ConfirmacionPeticion** y **SolicitudRespuesta**) utilizados, están definidos en el jar “scsp-beans”. Dichos objetos no son más que una representación de los elementos definidos por los esquemas SCSP. A diferencia de los esquemas SCSP, estos objetos son independientes de la versión de los esquemas SCSP: podremos utilizar los mismos objetos tanto para versiones V2 como para versiones V3. Es la configuración de las librerías la que se encargará de transformar dichos objetos a su representación SOAP durante el proceso de intercambio de mensajes con los emisor.

Por ejemplo, podríamos desarrollar un BackOffice de demostración muy sencillo, tal y como se muestra a continuación:

```
package es.scsp.services.test;
import java.io.StringWriter;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

import es.scsp.bean.common.Atributos;
import es.scsp.bean.common.ConfirmacionPeticion;
import es.scsp.bean.common.DatosGenericos;
import es.scsp.bean.common.Estado;
import es.scsp.bean.common.Peticion;
import es.scsp.bean.common.Respuesta;
import es.scsp.bean.common.Solicitante;
import es.scsp.bean.common.SolicitudTransmision;
import es.scsp.bean.common.Titular;
import es.scsp.bean.common.TransmisionDatos;
import es.scsp.bean.common.Transmisiones;
import es.scsp.common.backoffice.BackOffice;
import es.scsp.common.beans.RespuestaGenerator;
import es.scsp.common.exceptions.ScspException;
import es.scsp.common.exceptions.ScspExceptionConstants;
import es.scsp.services.ScspConstants;
```

/\*\*

\* Clase que simula un Backoffice real, este



\* Backoffice es para peticiones y respuestas de V2

\*

\* Los datos utilizados en esta clase son de prueba, para  
que sirva de guía a los desarrolladores.

\*/

```
public class DemoBackOfficeV2 implements BackOffice {

    private static final Log log = LogFactory.getLog(DemoBackOfficeV2.class);
    private static final String XMLNS = "http://intermediacion.redsara.es/scsp/esquemas/datosespecificos";

    public Respuesta NotificarSincrono(Peticion pet) throws ScspException {
        printDemo();
        log.debug("Procesando peticion sincrona.");
        try {
            ArrayList<TransmissionDatos> listaTransmissionDatos = new ArrayList<TransmissionDatos>();
            for (int i = 0; i < pet.getSolicitudes().getSolicitudTransmission().size(); i++) {
                SolicitudTransmission solicitudTransmission = pet.getSolicitudes().getSolicitudTransmission().get(i);
                DatosGenericos datosGenericosPeticion = solicitudTransmission.getDatosGenericos();
                Object datosEspecificos = solicitudTransmission.getDatosEspecificos();
                Solicitante datosSolicitanter = datosGenericosPeticion.getSolicitante();
                //Aqui se muestran los datos del solicitante de la peticion recibida.
                log.info("Datos del solicitante recibidos:");
                log.info(" - Identificador solicitante --> " + datosSolicitanter.getIdentificadorSolicitante());
                log.info(" - Nombre solicitante --> " + datosSolicitanter.getNombreSolicitante());
                log.info(" - Finalidad de la comunicacion --> " + datosSolicitanter.getFinalidad());
                Titular titular = datosGenericosPeticion.getTitular();
                log.info("Datos del titular recibidos:");
                log.info(" - Nombre completo del titular --> " + titular.getNombreCompleto());
                log.info(" - Tipo documentacion del titular --> " + titular.getTipoDocumentacion().name());
                log.info(" - Documentacion del titular --> " + titular.getDocumentacion());
                if (datosEspecificos != null) {
                    if (datosEspecificos instanceof Element) {
                        //Se muestran los datos especificos del resto de peticiones.
                        TransformerFactory transFactory = TransformerFactory.newInstance();
                        Transformer transformer = transFactory.newTransformer();
                        StringWriter buffer = new StringWriter();
                        transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
                        transformer.transform(new DOMSource((Element) datosEspecificos), new StreamResult(buffer));
                        log.info("Datos especificos (XML):" + buffer.toString());
                    } else {
                        log.info("Tipo de datos especificos desconocido.");
                    }
                } else {
                    log.info("La peticion no tiene datos especificos.");
                }
                //Generando la transmision de datos de la respuesta.
                TransmissionDatos transmissionDatos = new TransmissionDatos();
                transmissionDatos.setDatosGenericos(solicitudTransmission.getDatosGenericos());
                // Generando datos especificos de respuesta por cada solicitud de transmision.
                // Como es un backoffice de prueba la respuesta generada constara de un
                // documento xml donde aparecera el codigo del certificado del servicio, fecha de la
                // respuesta y una descripcion.
                transmissionDatos.setDatosEspecificos(generarDatosEspecificosRespuesta(datosGenericosPeticion.getTransmission().getCodigoCertificado()));
                listaTransmissionDatos.add(transmissionDatos);
            }
            Transmisiones transmisiones = new Transmisiones();
            transmisiones.setTransmissionDatos(listaTransmissionDatos);
            Respuesta respuesta = new Respuesta();
            respuesta.setAtributos(pet.getAtributos());
            respuesta.getAtributos().setEstado(new Estado());
            respuesta.getAtributos().getEstado().setCodigoEstado("0003");
            respuesta.getAtributos().getEstado().setCodigoEstadoSecundario("");
            respuesta.getAtributos().getEstado().setLiteralError("Peticion procesada correctamente.");
            respuesta.getAtributos().getEstado().setTiempoEstimadoRespuesta(0);
            respuesta.setTransmisiones(transmisiones);
            return respuesta;
        } catch (Exception ex) {
            String []a={""}

```

```
throw ScspException.getScspException(ScspExceptionConstants.ErrorBackOfficePruebas, "Error al procesar la
notificacion sincrona.");

}
}

public ConfirmacionPetition NotificarAsincrono(Petition pet) throws ScspException {
    printDemo();
    log.info("Procesando petition asincrona.");
    try {
        int tiempoRespuesta = 5;
        ConfirmacionPetition respuesta = new ConfirmacionPetition();
        respuesta.setAtributos(pet.getAtributos());
        respuesta.getAtributos().setEstado(new Estado());
        respuesta.getAtributos().getEstado().setCodigoEstado(ScspConstants.CodigoEstado_EnProceso);
        respuesta.getAtributos().getEstado().setCodigoEstadoSecundario("");
        respuesta.getAtributos().getEstado().setLiteralError("Petition procesada correctamente.");
        respuesta.getAtributos().getEstado().setTiempoEstimadoRespuesta(tiempoRespuesta);
        respuesta.getAtributos().setTimeStamp(pet.getAtributos().getTimeStamp());
        log.info("Devolviendo respuesta asincrona.");
        return respuesta;
    } catch (Exception e) {
        String []a={""}
    }
    throw ScspException.getScspException(ScspExceptionConstants.ErrorBackOfficePruebas, "Error al procesar la
notificacion sincrona.");

}
}

public Respuesta SolicitudRespuesta(es.scsp.bean.common.SolicitudRespuesta sr) throws ScspException {
    printDemo();
    log.info("Procesando solicitud de respuesta.");
    Atributos atributos = sr.getAtributos();
    Respuesta respuesta = new RespuestaGenerator().generateRespuesta(atributos);
    Object de = generarDatosEspecificosRespuesta(sr.getAtributos().getCodigoCertificado());
    respuesta.getTransmisiones().getTransmisionDatos().get(0).setDatosEspecificos(de);
    log.info("Generada respuesta de demostracion.");
    return respuesta;
}

/**
 * Metodo que genera los datos especificos de respuesta, estos datos son de prueba
 * y son un documento XML donde aparece el codigo de certificado, una descripcion
 * y la fecha en que se genera la respuesta.
 *
 * <DatosEspecificos xmlns="http://www.map.es/scsp/esquemas/datosespecificos">
 *   <descripcion>Respuesta de Backoffice V2 de prueba</descripcion>
 *   <codigoCertificado>XXXXXXXXXX</codigoCertificado>
 *   <fechaRespuesta>XXXXXXXXXX</fechaRespuesta>
 * </DatosEspecificos>
 *
 * @param codigoCertificado String
 * @return Element
 */
private Element generarDatosEspecificosRespuesta(String codigoCertificado) {
    DocumentBuilderFactory fac = DocumentBuilderFactory.newInstance();
    fac.setNamespaceAware(false);
    Document doc;
    try {
        doc = fac.newDocumentBuilder().newDocument();
    } catch (ParserConfigurationException e) {
        throw new RuntimeException("Error al crear el documento.", e);
    }
    Element result = doc.createElement("DatosEspecificos");
    // Generando elemento descripcion de la respuesta.
    Element descripcion = doc.createElement("descripcion");
    descripcion.setTextContent("Respuesta de Backoffice V2 de prueba");
    // Generando elemento codigoCertificado de la respuesta.
    Element codigoCertificadoElement = doc.createElement("codigoCertificado");
    if (codigoCertificado != null) {
```

```
descripcion.setTextContent(codigoCertificado);
} else {
    descripcion.setTextContent("No existe el codigo de certificado");
}
// Generando elemento fechaRespuesta de la respuesta.
Element fechaRespuesta = doc.createElement("fechaRespuesta");
fechaRespuesta.setTextContent(fechaDeHoy());
result.setAttribute("xmlns", XMLNS);
result.appendChild(descripcion);
result.appendChild(codigoCertificadoElement);
result.appendChild(fechaRespuesta);
return result;
}

/**
 * Metodo que devuelve la fecha actual en el formato dd/MM/yyyy HH:mm:ss:SSS
 * @return String
 */
private String fechaDeHoy() {
    Date date = new Date();
    String formato = new String("dd/MM/yyyy HH:mm:ss:SSS");
    SimpleDateFormat formatoSimple = new SimpleDateFormat(formato);
    return formatoSimple.format(date);
}

private void printDemo() {
    log.info("=====");
    log.info("BackOffice de demostracion");
    log.info("Este BackOffice no tiene otra finalidad que servir como");
    log.info("ejemplo para la realizacion de un BackOffice real.");
    log.info("=====");
}
}
```

Este BackOffice de pruebas simplemente imprime en el log los valores que recibe sin realizar ningún tratamiento de estos.

Vemos que implementa los tres métodos de la interface.

En el caso del método “**NotificarSincrono**” lo único que hace es llamar al método que ejecuta la lógica de negocio de pruebas para cada transmisión. Genera la respuesta a partir de los mismos datos que recibe, exceptuando los datos específicos, en los que simplemente devuelve un mensaje indicando si se encuentra el nodo CIF dentro de los datos específicos.

También podemos observar cómo, en este caso, los datos específicos se recuperan y envían como un objeto de tipo Element (org.w3c.dom.Element). De este modo, no necesitamos realizar ninguna acción para trabajar con diferentes esquemas de SCSP (por ejemplo podemos procesar tanto datos específicos de CDI como de la AEAT o la TGSS sin necesidad de codificar nada).

## 6.1 Integración de nuestro BackOffice con las librerías

Llegados a este punto, tenemos nuestra clase BackOffice encargada de realizar la lógica de negocio de nuestro emisor. ¿Y ahora qué?

El siguiente paso es integrar dicho BackOffice dentro de la aplicación web encargada de exponer los servicios web.

Para registrar nuestro BackOffice dentro de la aplicación deberemos registrar dicho BackOffice en la tabla de configuración del sistema **emisor\_backoffice**.

Dicha tabla contiene los siguientes campos:

- **idCertificado**: código del certificado con el que deseamos procesar los mensajes.
- **beanName**: alias del componente instalado en el contexto de Spring.

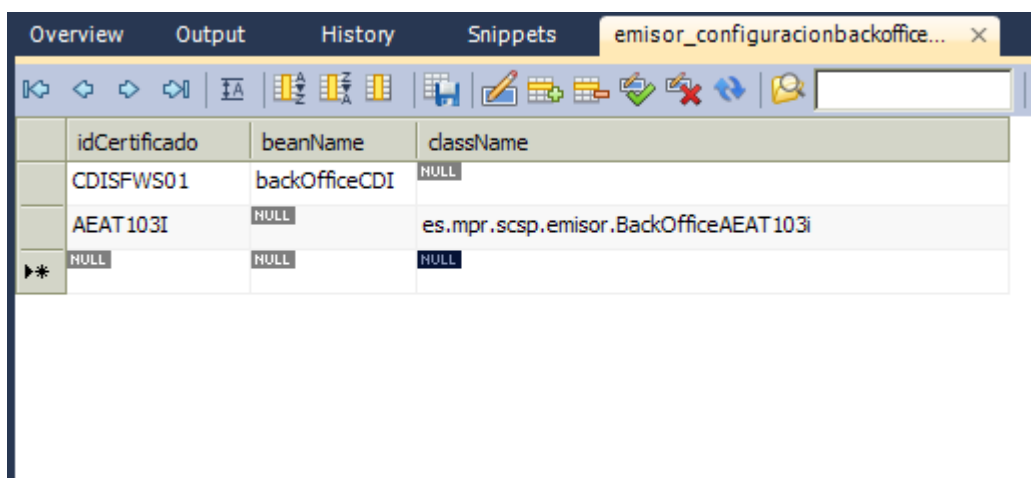
**\*Nota:** Esta propiedad sólo deberá utilizarse en el caso de integrar el BackOffice con el contexto Spring de las librerías.

- **className**: nombre completo de la clase del backoffice.

**\*Nota:** Esta propiedad tendrá como valor el nombre de la clase java que debe cumplir la siguiente estructura:

**<paquete>.<nombre\_de\_la\_clase>**

**Por ejemplo: es.mpr.scsp.emisor.BackOfficeAEAT103i**



idCertificado	beanName	className
CDISFWS01	backOfficeCDI	NULL
AEAT 103I	NULL	es.mpr.scsp.emisor.BackOfficeAEAT 103i
NULL	NULL	NULL

En este ejemplo de configuración utilizamos como BackOffice para CDI un BackOffice definido con el alias 'backOfficeCDI' dentro del contexto de Spring. Para el caso del certificado AEAT103I, especificamos simplemente el nombre de la clase (obviamente dicha clase deberá añadirse al classpath de nuestra aplicación).

De este modo, podemos registrar un determinado BackOffice para cada uno de los certificados instalados en el sistema.

Como mencionábamos en el punto anterior, podemos hacer referencia a nuestro BackOffice, bien a través del nombre de Spring, o bien a través del nombre de la clase.

*¿Qué diferencias hay por lo tanto, y qué ventajas tiene cada sistema?*

Utilizando la opción de Spring podemos fácilmente hacer que nuestro BackOffice utilice otros recursos del sistema. No vamos aquí a enumerar las bondades de Spring, basta decir que instanciando de este modo nuestro bean podríamos definir de forma sencilla aquellos componentes necesarios para nuestro procesamiento. Si, por ejemplo, nuestra implementación

tuviera que acceder a un servicio para validar los datos, y a otro servicio para registrar la información solicitada, podríamos hacerlo de un modo similar al siguiente:

```
<bean id="myBackOffice" class="es.scsp.services.emisor.MyBackOfficeDeEjemplo">
  <property name="validationService" ref=" validationService " />
  <property name="registrationService" ref=" registrationService " />
</bean>

<bean id=" validationService " class="es.scsp.services.emisor.ValidationService">
  <property name="validationDao" ref="validationDao" />
</bean>

<bean id=" validationDao " class="es.scsp.services.emisor.ValidationDao">
  <property name="dataSource" ref="dataSource" />
</bean>
```

De este modo es más sencillo organizar nuestro desarrollo a la hora de integrar cada uno de los componentes del sistema. En este ejemplo, el servicio de validación utiliza un DAO definido en el propio contexto que a su vez utiliza una conexión con base de datos.

Bastaría con ir añadiendo al fichero de configuración de Spring (applicationContext.xml) aquellos componentes que deban ser utilizados por nuestra lógica de negocio.

Aparte de utilizar la integración, a través del contexto de configuración podemos prescindir de este sistema, e indicar únicamente el nombre de la clase. En este caso, la instancia de BackOffice se generará **a través de reflexión**, utilizando para ello el constructor sin argumentos de la clase (obligatoriamente nuestro BackOffice ha de tener un constructor sin argumentos).

**\*Nota:** Obligatoriamente el backoffice que se defina debe tener un constructor sin argumentos.

Este sistema es menos recomendable, ya que es el BackOffice el encargado de crear el resto de instancias de aquellos componentes:

Por ejemplo, en el caso anterior sería el constructor (o cualquier otro método de inicialización) de la clase el encargado de crear las instancias de los servicios que precisa, algo que aumenta el acoplamiento entre componentes al mismo tiempo que exige más código para realizar las mismas operaciones.

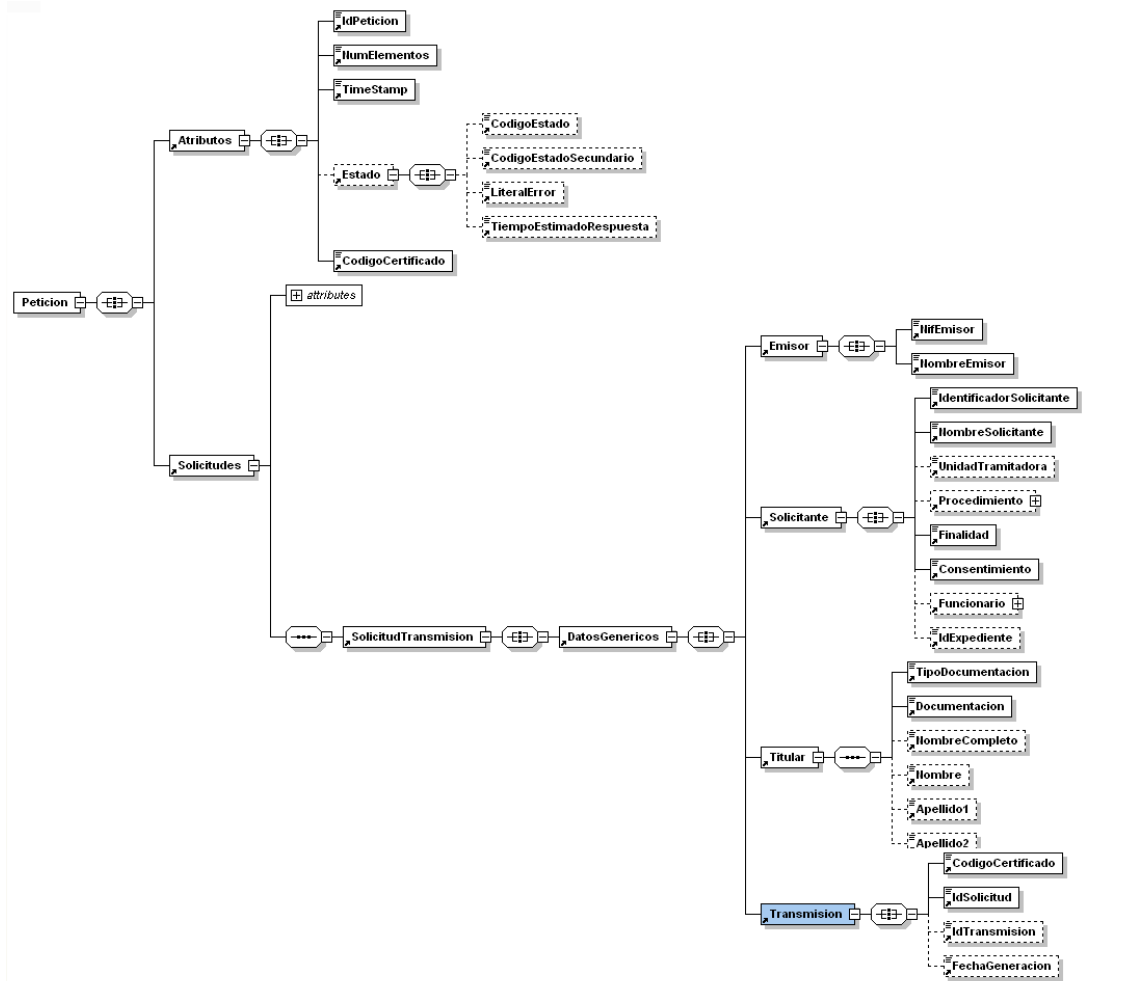
## 7 Apéndice

### 7.1 Emisor que simula el servicio AEAT103I

Dentro de la distribución se proporciona un emisor de prueba que simula el funcionamiento un emisor real de la AEAT concretamente el del servicio AEAT103I. En las siguientes líneas se hablará de sus características, como programarlo y como poder realizar las pruebas.

#### 7.1.1 Características

Es un emisor que responde a **peticiones síncronas** con el siguiente esquema:

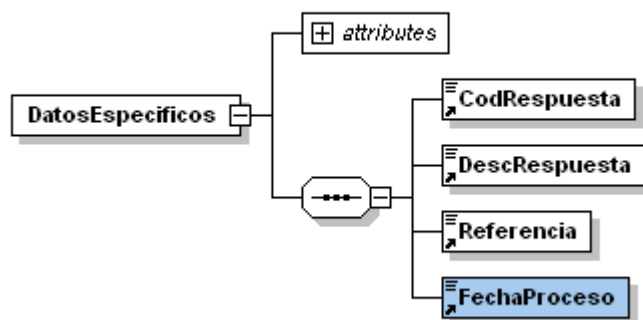


Este servicio **es un servicio de esquema V3**, que dependiendo del titular que viaje en la petición, no indicará si dicha persona está al corriente de pagos o no, este servicio puede devolver 4 tipos de código, que a su vez se puede subdividir en dos grupos uno positivo el código 0, el titular está al corriente de pagos y otros 3 negativos que de forma general indican el que el titular no está al corriente de pagos, de una forma más concreta sería la siguiente:

- Código 0 → El usuario está al corriente de pago, la descripción que envía el servicio sería la siguiente: "El contribuyente se encuentra al corriente en sus obligaciones tributarias".

- **Código 1** → El usuario **NO** está al corriente de pago, la descripción que envía el servicio sería la siguiente: "El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado que no figura presentación de declaración o de autoliquidación a la que figura obligado".
- **Código 2** → El usuario **NO** está al corriente de pago, la descripción que envía el servicio sería la siguiente: "El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado la existencia de una deuda o sanción de carácter tributario"
- **Código 3** → El usuario **NO** está al corriente de pago, la descripción que envía el servicio sería la siguiente: "El contribuyente no se encuentra al corriente en sus obligaciones tributarias debido a cualquier otra causa o combinación de ellas"

Sólo uno de estos códigos podrá aparecer en la respuesta del servicio, concretamente en el nodo de Datos Específicos que tiene la siguiente estructura:



La descripción de estos campos sería la siguiente:

Nodo	Descripción
<b>CodRespuesta</b>	Código de respuesta del servicio puede tomar los valores 0,1,2,3
<b>DescRespuesta</b>	Descripción asociado al código que devuelva el servicio.
<b>Referencia</b>	Cadena alfanumérica asociada al contribuyente.
<b>FechaProceso</b>	Fecha en la que se realiza la petición, con formato (yyyy-MM-dd)

Un ejemplo de respuesta podría ser la siguiente:

```

<?xml version="1.0" encoding="UTF-8"?>
<Respuesta xmlns="http://intermediacion.redsara.es/scsp/esquemas/ws/respuesta">
  <Atributos>
    <IdPetition>TESTBRK30000066V</IdPetition>
    <NumElementos>1</NumElementos>
    <TimeStamp>2011-03-30T14:03:07.128+02:00</TimeStamp>
    <Estado>
      <CodigoEstado>0003</CodigoEstado>
      <CodigoEstadoSecundario/>
      <LiteralError>Petición procesada correctamente.</LiteralError>
      <TiempoEstimadoRespuesta>0</TiempoEstimadoRespuesta>
    </Estado>
  </Atributos>
</Respuesta>
  
```

```
<CodigoCertificado>AEAT103I</CodigoCertificado>
</Atributos>
<Transmisiones>
  <TransmisionDatos>
    <DatosGenericos>
      <Emisor>
        <NifEmisor>Q2826000H</NifEmisor>
        <NombreEmisor>AEAT</NombreEmisor>
      </Emisor>
      <Solicitante>
        <IdentificadorSolicitante>Q2827003A</IdentificadorSolicitante>
        <NombreSolicitante>ALTEN</NombreSolicitante>
        <Finalidad>PRUEBAS</Finalidad>
        <Consentimiento>Si</Consentimiento>
      </Solicitante>
      <Titular>
        <TipoDocumentacion>NIF</TipoDocumentacion>
        <Documentacion>47046991</Documentacion>
      </Titular>
      <Transmision>
        <CodigoCertificado>AEAT103I</CodigoCertificado>
        <IdSolicitud>TESTBRK30000066V</IdSolicitud>
        <IdTransmision>TESTBRK30000066V</IdTransmision>
        <FechaGeneracion>2011-03-
30T14:03:07.128+02:00</FechaGeneracion>
      </Transmision>
    </DatosGenericos>
    <DatosEspecificos>
      xmlns="http://intermediacion.redsara.es/scsp/esquemas/datosespecificos">
        <CodRespuesta xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">2</CodRespuesta>
        <DescRespuesta>El contribuyente no se encuentra al corriente en sus
obligaciones tributarias, se ha detectado la existencia de una deuda o sanción de carácter tributario</DescRespuesta>
        <Referencia>71113000810</Referencia>
        <FechaProceso>30-03-2011</FechaProceso>
      </DatosEspecificos>
    </TransmisionDatos>
  </Transmisiones>
</Respuesta>
```

**\*\*Nota:** Para más información ver el documento de integración Servicio de Verificación de Estar al Corriente de Pagos para Becas y Subvenciones que se encuentra en <http://administracionelectronica.gob.es/es/ctt/svd>

### 7.1.2 Programación y simulación del emisor AEAT103I

Para desarrollar y simular su comportamiento nos basaremos en un fichero de propiedades, denominado *scsp-aeat-demo.properties*, que tiene la siguiente estructura:

```
nif.XXXXXXXXXX=true
nif.ZZZZZZZZZ=false
nif.YYYYYYYYYY=true
nif.777777777=true
nif.888888888=false
nif.999999999=false
```

```
msg.nocorriente.0=El contribuyente se encuentra al corriente en sus obligaciones tributarias
msg.nocorriente.1=El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado que no
figura presentación de declaración o de autoliquidación a la que figura obligado
msg.nocorriente.2=El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado la
existencia de una deuda o sanción de carácter tributario
msg.nocorriente.3=El contribuyente no se encuentra al corriente en sus obligaciones tributarias debido a cualquier otra
causa o combinación de ellas
```



En este archivo podemos ver dos patrones de propiedades diferentes:

- `nif.[número_de_la_documentacion]` → Estas propiedades tienen asociados dos tipos de valores `true/false` que nos indica si el nif de la persona sobre la que se está consultando, está o no al corriente de pagos, entonces si la propiedad tiene valor `true`, el usuario estará al corriente de pago, si no el usuario no estará al corriente de pago. Puede haber tantas propiedades con este formato como se quiera, un ejemplo sería el siguiente:

```
nif.XXXXXXXXXX=true  
nif.YYYYYYYY=false
```

El valor que se encuentra entre corchetes ha de sustituirse por el número de la documentación que se le quiere asociar si está al corriente de pago o no.

- `msg.nocorriente.[código]` → Sólo puede haber 4 propiedades con este formato ya que el servicio de la AEAT103I sólo devuelve cuatro tipos de código 0,1,2 y 3, que tienen asociada su descripción correspondiente, entonces los cuatro propiedades que pueden aparecer son los siguientes:

```
msg.nocorriente.0=El contribuyente se encuentra al corriente en sus obligaciones tributarias  
msg.nocorriente.1=El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado que no  
figura presentación de declaración o de autoliquidación a la que figura obligado  
msg.nocorriente.2=El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado la  
existencia de una deuda o sanción de carácter tributario  
msg.nocorriente.3=El contribuyente no se encuentra al corriente en sus obligaciones tributarias debido a cualquier otra  
causa o combinación de ellas
```

En caso de que se añadiera un nuevo código simplemente hay que añadir una nueva propiedad con el formato indicado anteriormente.

Una vez que se tiene definido el fichero de propiedades que se ubicará en la carpeta source para que tenga acceso desde el classpath del emisor, se debe crear el backoffice de prueba siguiendo los pasos indicados “[en el punto 6 Creación del BackOffice](#)”

La diferencia con los pasos indicados en la creación del backoffice radica en la forma de generar los datos específicos, que seguirá siguiendo un documento XML pero la forma de generar sus nodos se deberá hacer teniendo en cuenta el esquema de datos específicos ([véase el punto 7.1.1 Características](#)), un posible código que genera los datos específicos según el esquema sería el siguiente:

```
boolean estaAlcorriente = estaAlcorrienteDePago(titular.getDocumentacion());  
String descripcion = descripcionAlcorriente(estaAlcorriente);  
  
...  
  
TransmisionDatos datos = new TransmisionDatos();  
datos.setDatosGenericos(datosGenericosPeticion);  
datos.setDatosEspecificos(generateDatosEspecificos(descripcion));  
  
private boolean estaAlcorrienteDePago(String nif)throws Exception{  
  
    boolean estaAlcorriente= true;  
    try {  
        estaAlcorriente = Boolean.parseBoolean(properties.getProperty("nif."+nif));  
    }catch(Exception ex){  
        estaAlcorriente=false;  
    }  
  
    return estaAlcorriente;  
}
```

```
private String descripcionAlcorriente(boolean estaAlcorriente){
    String descripcion = properties.getProperty("msg.nocorriente."+0);
    int numMsg = 0;
    if(!estaAlcorriente){
        Random rand = new Random();
        numMsg = rand.nextInt(3)+1;
        descripcion = properties.getProperty("msg.nocorriente."+numMsg);
    }

    return descripcion+"-"+numMsg;
}

private Element generateDatosEspecificos(String descripcion) throws ParserConfigurationException{

    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setNamespaceAware(true);

    Date fecha = new Date();
    SimpleDateFormat formatoDeFecha = new SimpleDateFormat("dd-MM-yyyy");

    String[] descripcionCodigo = descripcion.split("-");

    Document tmp = factory.newDocumentBuilder().newDocument();

    Element datosEspecificos = tmp.createElement("DatosEspecificos");

    Element codRespuesta = tmp.createElement("CodRespuesta");
    codRespuesta.appendChild(tmp.createTextNode(descripcionCodigo[1]));
    Element descRespuesta = tmp.createElement("DescRespuesta");
    descRespuesta.appendChild(tmp.createTextNode(descripcionCodigo[0]));
    Element referencia = tmp.createElement("Referencia");
    referencia.appendChild(tmp.createTextNode(REFERENCIA));
    Element fechaProceso = tmp.createElement("FechaProceso");
    fechaProceso.appendChild(tmp.createTextNode(formatoDeFecha.format(fecha)));

    datosEspecificos.appendChild(codRespuesta);
    datosEspecificos.appendChild(descRespuesta);
    datosEspecificos.appendChild(referencia);
    datosEspecificos.appendChild(fechaProceso);
    datosEspecificos.setAttribute("xmlns", XMLNS);

    tmp.appendChild(datosEspecificos);

    try{
        TransformerFactory tranFactory = TransformerFactory.newInstance();
        Transformer aTransformer = tranFactory.newTransformer();

        Source src = new DOMSource(tmp);
        Result dest = new StreamResult(System.out);
        aTransformer.transform(src, dest);
    }catch(Exception ex){
        ex.printStackTrace();
    }

    return tmp.getDocumentElement();
}
```

El funcionamiento de estos métodos es el siguiente:

- **boolean estaAlcorrienteDePago(String nif)** → Este método se encarga de ir al archivo de propiedades y dependiendo del nif que se le pase por parámetro devolverá un valor true o false, para ello lo que hace es concatenar la cadena "nif." con el número de nif que se le pasa por parámetro y con ese valor de propiedad se coge el valor correspondiente, en caso de meter un nif inexistente, es decir, que concatenando "nif." más el número de nif pasado por parámetro no existiese esa propiedad se devolverá valor false, en caso contrario el que tenga asociado.

- **String descripcionAlcorriente(boolean estaAlcorriente)** → Este método se encarga de devolver la descripción dependiendo si el titular está al corriente de pagos o no, y esto se hará de la siguiente forma:
  - **estaAlcorriente == true** → El usuario está al corriente de pago por lo que se devuelve un código de respuesta 0 y como descripción que el titular está al corriente de pago, cogiendo lo del archivo de propiedades indicado anteriormente, para ello se concatena la cadena "msg.nocorriente" con el dígito 0.
  - **estaAlcorriente == false** → El usuario no está al corriente de pago, entonces como hay tres códigos negativos, es decir asociados a no estar al corriente de pagos, se coge un número aleatorio entre 0 y 2, que se le suma uno para que entre dentro del rango de valores de códigos de no estar al corriente de pagos, una vez se tiene este número se concatena con la cadena "msg.nocorriente" y se obtiene una cadena con la descripción-código.
- **Element generateDatosEspecificos(String descripcion)** → Este método se encarga de generar un documento XML con la estructura acorde con los datos específicos, entonces a este método se le pasa una cadena que tiene el siguiente formato "descripcion-codigo" entonces lo que se hace es separa la descripción y el código haciendo un split por la cadena "-" que nos devuelve un array de dos posiciones donde la primera posición (0) viene la descripción y en la segunda posición (1) con el código de respuesta, entonces cada valor se añade al nodo correspondiente generándose un objeto DOM que posteriormente se añade a la respuesta que se está generando, para más información sobre la composición de los datos específicos de este servicio "véase el punto 7.1.1 Características".

### 7.1.3 Como probar la simulación del servicio.

Para probar la ejecución de este servicio en la petición en la documentación del titular tendrá que viajar alguno de los definidos en el archivo "scsp-aeat-demo.properties", entonces en el emisor que se entrega en la distribución en dicho archivo vienen definidos los siguientes dni's:

NIF	RESPUESTA
XXXXXXXXXX	<p>El titular <b>SÍ</b> está al corriente de pago.</p> <p><b>Código de respuesta</b>→ 0</p> <p><b>Descripción de respuesta</b>→ <u>El contribuyente se encuentra al corriente en sus obligaciones tributarias.</u></p>
ZZZZZZZZZZ	<p>El titular <b>NO</b> está al corriente de pago.</p> <p><b>Código de respuesta</b>→ 1 o 2 o 3</p> <p><b>Descripción de respuesta</b>→ <u>El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado que no figura presentación de declaración o de autoliquidación a la que figura obligado o</u></p> <p><u>El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado la existencia de una deuda o sanción de carácter tributario o</u></p> <p><u>El contribuyente no se encuentra al corriente en sus</u></p>

	<u>obligaciones tributarias debido a cualquier otra causa o combinación de ellas</u>
YYYYYYYYYY	El titular <b>SI</b> está al corriente de pago. <b>Código de respuesta</b> → 0 <b>Descripción de respuesta</b> → <u>El contribuyente se encuentra al corriente en sus obligaciones tributarias.</u>
777777777	El titular <b>SI</b> está al corriente de pago. <b>Código de respuesta</b> → 0 <b>Descripción de respuesta</b> → <u>El contribuyente se encuentra al corriente en sus obligaciones tributarias.</u>
888888888	El titular <b>NO</b> está al corriente de pago. <b>Código de respuesta</b> → 1 o 2 o 3 <b>Descripción de respuesta</b> → <u>El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado que no figura presentación de declaración o de autoliquidación a la que figura obligado o</u> <u>El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado la existencia de una deuda o sanción de carácter tributario o</u> <u>El contribuyente no se encuentra al corriente en sus obligaciones tributarias debido a cualquier otra causa o combinación de ellas</u>
999999999	El titular <b>NO</b> está al corriente de pago. <b>Código de respuesta</b> → 1 o 2 o 3 <b>Descripción de respuesta</b> → <u>El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado que no figura presentación de declaración o de autoliquidación a la que figura obligado o</u> <u>El contribuyente no se encuentra al corriente en sus obligaciones tributarias, se ha detectado la existencia de una deuda o sanción de carácter tributario o</u> <u>El contribuyente no se encuentra al corriente en sus obligaciones tributarias debido a cualquier otra causa o combinación de ellas</u>

En el caso de que se meta un NIF que no corresponda con los que hay en la tabla anterior, se considera que no está al corriente de pago.

Si se quisiera añadir un nuevo nif al documento "scsp-aeat-demo.properties", se deberán seguir los siguientes pasos:

1. Abrir el documento scsp-aeat-demo.properties.

2. Añadir un nuevo registro con el siguiente formato:  
nif.[numero\_de\_nif\_nuevo]=[true/false]

Donde pone [numero\_de\_nif\_nuevo] se pondrá el nuevo número de nif y donde pone [true/false], se pondrá uno de los dos valores.

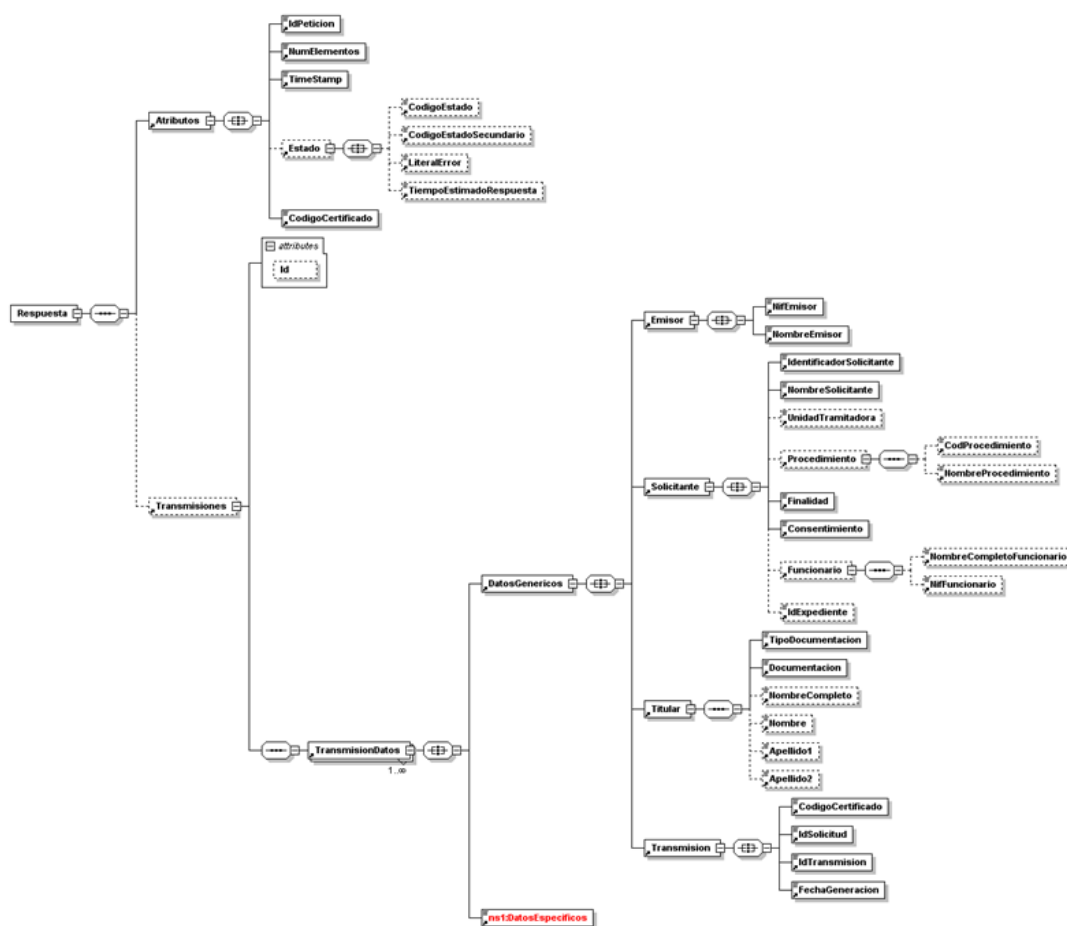
3. Guardar los cambios.

## 7.2 Emisor que simula el servicio Q2827003ATGSS006

Dentro de la distribución se proporciona un emisor de prueba que simula el funcionamiento un emisor real de la TGSS concretamente el del servicio Q2827003ATGSS006. En las siguientes líneas se hablará de sus características, como programarlo y como poder realizar las pruebas.

### 7.2.1 Características

Es un emisor que responde a **peticiones síncronas** con el siguiente esquema:

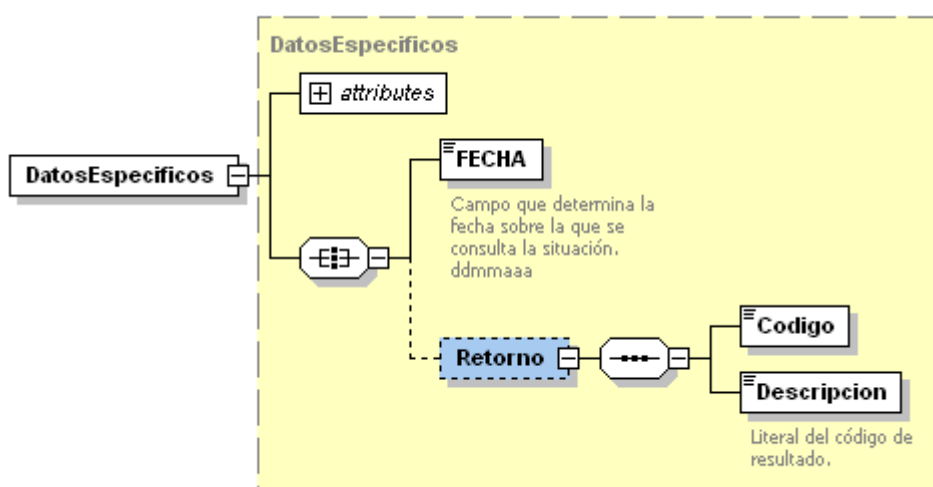


Este servicio **es un servicio de esquema V3**, que dependiendo de la fecha que viaje en el nodo de los datos específicos, se puede devolver alguno de los siguientes valores:

- **Código 0001** → El usuario está en situación de alta laboral, y el mensaje que se mostraría es el siguiente: *"En situación de Alta laboral a la fecha"*.
- **Código 0002** → El usuario **NO** está en situación de alta laboral, la descripción que envía el servicio sería la siguiente: *"No figura de Alta laboral a la fecha"*.
- **Código 0003** → El usuario **NO** está en situación de alta laboral, y el identificador de persona física no es correcto, con lo que la descripción: *"IPF erróneo"*
- **Código 0004** → El usuario **NO** está dado de alta laboral, y además el identificador de persona física no existe en la base de datos de la Seguridad Social, entonces la descripción que envía el servicio sería la siguiente: *"IPF inexistente en la base de datos de Seguridad Social"*

- Código 0005 → El usuario **NO** está dado de alta laboral, y además el identificador de persona física está duplicado en la base de datos de la Seguridad Social, entonces la descripción que envía el servicio sería la siguiente: *"El IPF está duplicado en la base de datos de Seguridad Social"*
- Código 0006 → No se ha podido comprobar si el usuario está dado de alta laboral porque el formato de fecha que viaja en los datos específicos no cumple con el formato correcto "ddMMyyyy", donde la descripción enviada sería la siguiente: *"El formato de la fecha de petición es incorrecto"*.

Sólo uno de estos códigos podrá aparecer en la respuesta del servicio, concretamente en el nodo de Datos Específicos que tiene la siguiente estructura:



La descripción de estos campos sería la siguiente:

Nodo	Descripción
<b>FECHA</b>	Fecha en formato ddMMyyyy
<b>Retorno/Codigo</b>	Código de respuesta del servicio puede tomar los valores 0001, 0002, 0003, 0004, 0005, y 0006
<b>Retorno/Descripcion</b>	Descripción asociado al código que devuelva el servicio.

Un ejemplo de respuesta podría ser la siguiente:

```

<?xml version="1.0" encoding="UTF-8"?>
<Respuesta xmlns="http://intermediacion.redsara.es/scsp/esquemas/V3/respuesta">
  <Atributos>
    <IdPetition>3cb28d8c00005592</IdPetition>
    <NumElementos>1</NumElementos>
    <TimeStamp>2011-02-21T15:51:56.937+01:00</TimeStamp>
    <Estado>
      <CodigoEstado>0003</CodigoEstado>
      <LiteralError>Solicitud de Certificado procesada correctamente</LiteralError>
    </Estado>
    <CodigoCertificado>Q2827003ATGSS006</CodigoCertificado>
  </Atributos>
  <Transmisiones>
    <TransmisionDatos>
      <DatosGenericos>
    </DatosGenericos>
  </Transmisiones>
</Respuesta>
  
```

```

Social</NombreEmisor>
    <Emisor>
        <NifEmisor>Q2827003A</NifEmisor>
        <NombreEmisor>Seguridad
    </Emisor>
    <Solicitante>
        <IdentificadorSolicitante>S2811001C</IdentificadorSolicitante>
        <NombreSolicitante>MPR</NombreSolicitante>
        <UnidadTramitadora>PRUEBA</UnidadTramitadora>
        <Procedimiento>
        <Finalidad>Prueba de consulta de datos corriente
de Pago con la Seguridad Social Alta en fecha (Q2827003ATGSS006)</Finalidad>
        <Consentimiento>Si</Consentimiento>
        <Funcionario>
            <NombreCompletoFuncionario>ANGEL
JAVIER SALIDO MARTINEZ</NombreCompletoFuncionario>
            <NifFuncionario>47046991P</NifFuncionario>
            <Funcionario>
            <Solicitante>
            <Titular>
                <TipoDocumentacion>NIF</TipoDocumentacion>
                <Documentacion>47046991P</Documentacion>
                <NombreCompleto>Angel Javier Salido
Martinez</NombreCompleto>
            <Titular>
            <Transmision>
                <CodigoCertificado>Q2827003ATGSS006</CodigoCertificado>
                <IdSolicitud>3cb28d8c00005592</IdSolicitud>
                <IdTransmision>T3cb28d8c00005592</IdTransmision>
                <FechaGeneracion>2011-02-
21</FechaGeneracion>
            </Transmision>
            <DatosGenericos>
            <DatosEspecificos>
xmlns="http://intermediacion.redsara.es/scsp/esquemas/datosespecificos" Id="Cifrado0">
                <FECHA>20022009</FECHA>
                <Retorno>
                    <Codigo>0004</Codigo>
                    <Descripcion>IPF inexistente en la base de datos
de Seguridad Social</Descripcion>
                </Retorno>
            </DatosEspecificos>
            </TransmisionDatos>
            </Transmisiones>
        </Respuesta>

```

**\*\*Nota:** Para más información ver el documento de integración Servicio de Verificación de Estar al Corriente de Pagos para Becas y Subvenciones que se encuentra en <http://administracionelectronica.gob.es/es/ctt/svd>

## 7.2.2 Programación y simulación del emisor Q2827003ATGSS006

Para desarrollar y simular su comportamiento nos basaremos en un fichero de propiedades, denominado *scsp-tgss-af-demo.properties*, que tiene la siguiente estructura:

```

tgss.fechaAlta=25022000
tgss.fechaBaja=15022011

tgss.0001.msg=En situación de Alta laboral a la fecha
tgss.0002.msg=No figura de Alta laboral a la fecha

```



```
tgss.0003.msg=IPF erróneo
tgss.0004.msg=IPF inexistente en la base de datos de Seguridad Social
tgss.0005.msg=El IPF está duplicado en la base de datos de Seguridad Social
tgss.0006.msg=El formato de la fecha de petición es incorrecto
```

En este archivo podemos ver dos patrones de propiedades diferentes:

- tgss.fechaAlta → Esta propiedad indica la fecha en la que el usuario con el que se está simulando se dio de alta en la Seguridad Social. Este campo tiene como formato ddMMyyyy y **debe tener algún valor**.
- tgss.fechaBaja → Esta propiedad indica la fecha en la que el usuario con el que se está simulando se dio de baja en la Seguridad Social. Este campo tiene como formato ddMMyyyy y **debe tener algún valor superior al de alta**.

Un ejemplo de los dos puntos anteriormente explicados sería el siguiente:

```
tgss.fechaAlta=25022000
tgss.fechaBaja=15022011
```

- tgss.[código\_mensaje].msg → Estas propiedades en un principio sólo pueden ser seis, que se corresponden con cada uno de los códigos de respuesta que puede generar el servicio Q2827003ATGSS006, entonces las propiedades que pueden aparecer son las siguientes:

```
tgss.0001.msg=En situación de Alta laboral a la fecha
tgss.0002.msg=No figura de Alta laboral a la fecha
tgss.0003.msg=IPF erróneo
tgss.0004.msg=IPF inexistente en la base de datos de Seguridad Social
tgss.0005.msg=El IPF está duplicado en la base de datos de Seguridad Social
tgss.0006.msg=El formato de la fecha de petición es incorrecto
```

En caso de que se añadiera un nuevo código simplemente hay que añadir una nueva propiedad con el formato indicado anteriormente.

Una vez que se tiene definido el fichero de propiedades que se ubicará en la carpeta source para que tenga acceso desde el classpath del emisor, se debe crear el backoffice de prueba siguiendo los pasos indicados “en el punto 6 Creación del BackOffice”

La diferencia con los pasos indicados en la creación del backoffice radica en la forma de generar los datos específicos, que seguirá siguiendo un documento XML pero la forma de generar sus nodos se deberá hacer teniendo en cuenta el esquema de datos específicos ([véase el punto 7.2.1 Características](#)), un posible código que genera los datos específicos según el esquema sería el siguiente:

```
private Element generarDatosEspecificos(String codigo, String descripcion, String fecha) throws
ParserConfigurationException{

    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setNamespaceAware(true);

    Document tmp = factory.newDocumentBuilder().newDocument();

    Element datosEspecificos = tmp.createElement("DatosEspecificos");
    Element nodoFecha = tmp.createElement("FECHA");
    Element retorno = tmp.createElement("Retorno");

    nodoFecha.appendChild(tmp.createTextNode(fecha));
    Element codigoNodo = tmp.createElement("Codigo");
    codigoNodo.appendChild(tmp.createTextNode(codigo));
```

```
Element descripcionNodo = tmp.createElement("Descripcion");
descripcionNodo.appendChild(tmp.createTextNode(descripcion));

retorno.appendChild(codigoNodo);
retorno.appendChild(descripcionNodo);
datosEspecificos.appendChild(nodoFecha);
datosEspecificos.appendChild(retorno);

datosEspecificos.setAttribute("xmlns", XMLNS);

tmp.appendChild(datosEspecificos);

try{
    TransformerFactory tranFactory = TransformerFactory.newInstance();
    Transformer aTransformer = tranFactory.newTransformer();

    Source src = new DOMSource(tmp);
    Result dest = new StreamResult(System.out);
    aTransformer.transform(src, dest);
}catch(Exception ex){
    ex.printStackTrace();
}
return tmp.getDocumentElement();
}
```

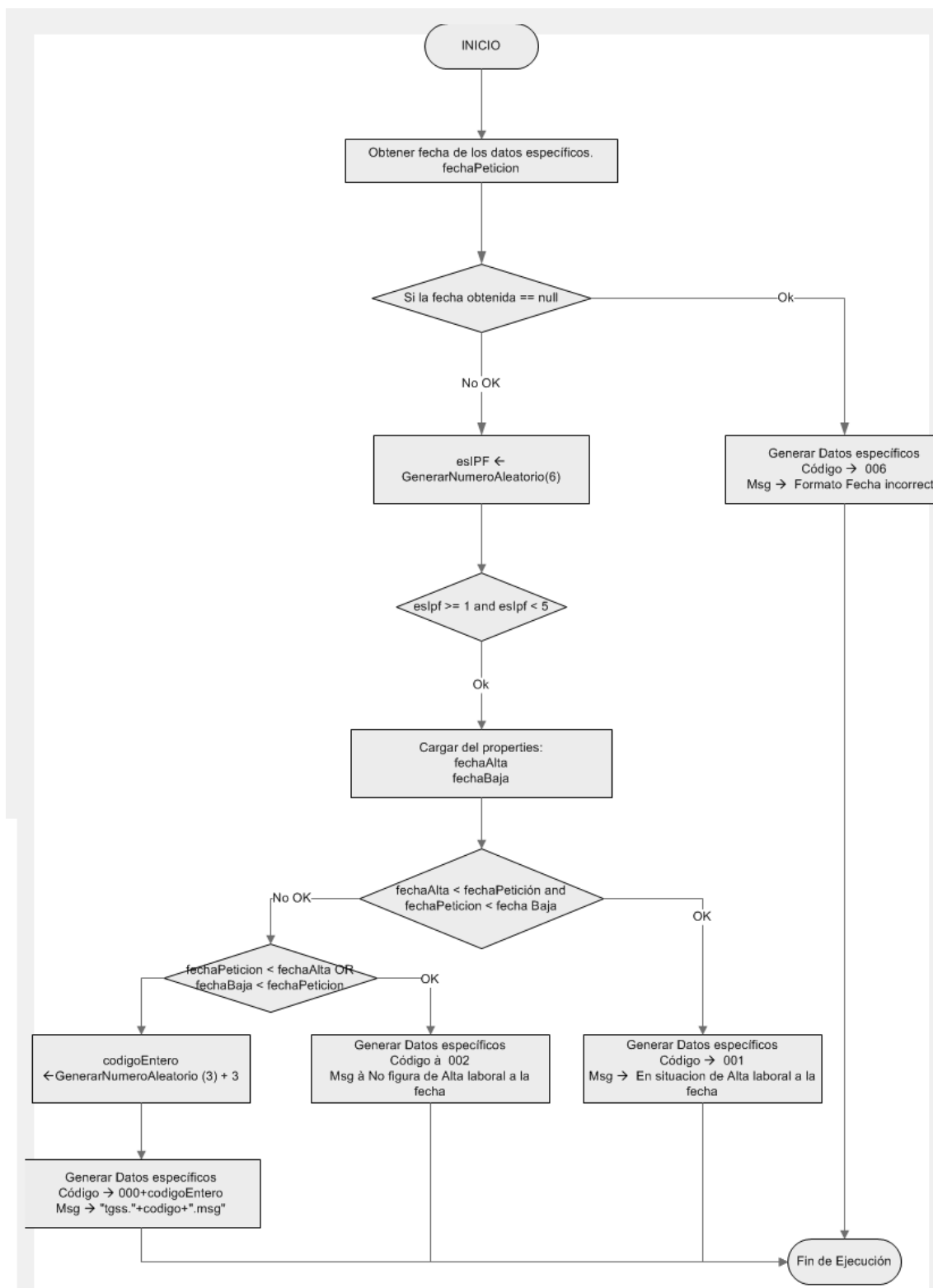
El funcionamiento de este método es el siguiente:

- **private** Element generarDatosEspecificos(String codigo, String descripcion, String fecha) → Este método se encarga de la generación del documento XML con formato de los datos específicos que luego se instará dentro de la respuesta que se envíe al cliente, este método recibe como parámetro los siguientes:
  - código → Valor que debe estar comprendido entre 0001 y 0006 que son los únicos valores que pueden ser devueltos por el servicio.
  - descripción → Es la descripción asociada al código que se está devolviendo.
  - Fecha → Es la fecha enviada en los datos específicos de la petición.

Este método devuelve un documento XML como el siguiente:

```
<DatosEspecificos xmlns="http://intermediacion.redsara.es/scsp/esquemas/datosespecificos">
  <FECHA xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">15032011</FECHA>
  <Retorno>
    <Codigo>0005</Codigo>
    <Descripcion>El IPF está duplicado en la base de datos de Seguridad Social</Descripcion>
  </Retorno>
</DatosEspecificos>
```

La elección de la descripción dependerá del siguiente algoritmo:



### 7.2.3 Como probar la simulación del servicio.

Para probar la ejecución de este servicio en los datos específicos de la petición, deberá venir una fecha que cumpla alguno de los siguientes casos:

FECHA	RESPUESTA
FECHA_PET > FECHA_ALTA && FECHA_PET < FECHA_BAJA	El titular <b>SI</b> está dado de alta en la Seguridad Social <b>Código de respuesta</b> → 0001 <b>Descripción de respuesta</b> → <u>En situación de Alta laboral a la fecha.</u>
FECHA_PET < FECHA_ALTA OR FECHA_PET > FECHA_BAJA	El titular <b>NO</b> está dado de alta en la Seguridad Social. <b>Código de respuesta</b> → 0002 <b>Descripción de respuesta</b> → <u>No figura de Alta laboral a la fecha</u>
FECHA_PET == NULL	El titular <b>SI</b> está al corriente de pago. <b>Código de respuesta</b> → 0006 <b>Descripción de respuesta</b> → <u>El formato de la fecha de petición es incorrecto.</u>
NINGUNO DE LOS CASOS ANTERIORES	El usuario puede recibir alguna de las siguientes respuestas. <b>Código de respuesta</b> → 0003, 0004 o 0005 <b>Descripción de respuesta:</b> <b>0003</b> → <u>IPF erróneo.</u> <b>0004</b> → <u>IPF inexistente en la base de datos de Seguridad Social</u> <b>0005</b> → <u>El IPF está duplicado en la base de datos de Seguridad Social</u>

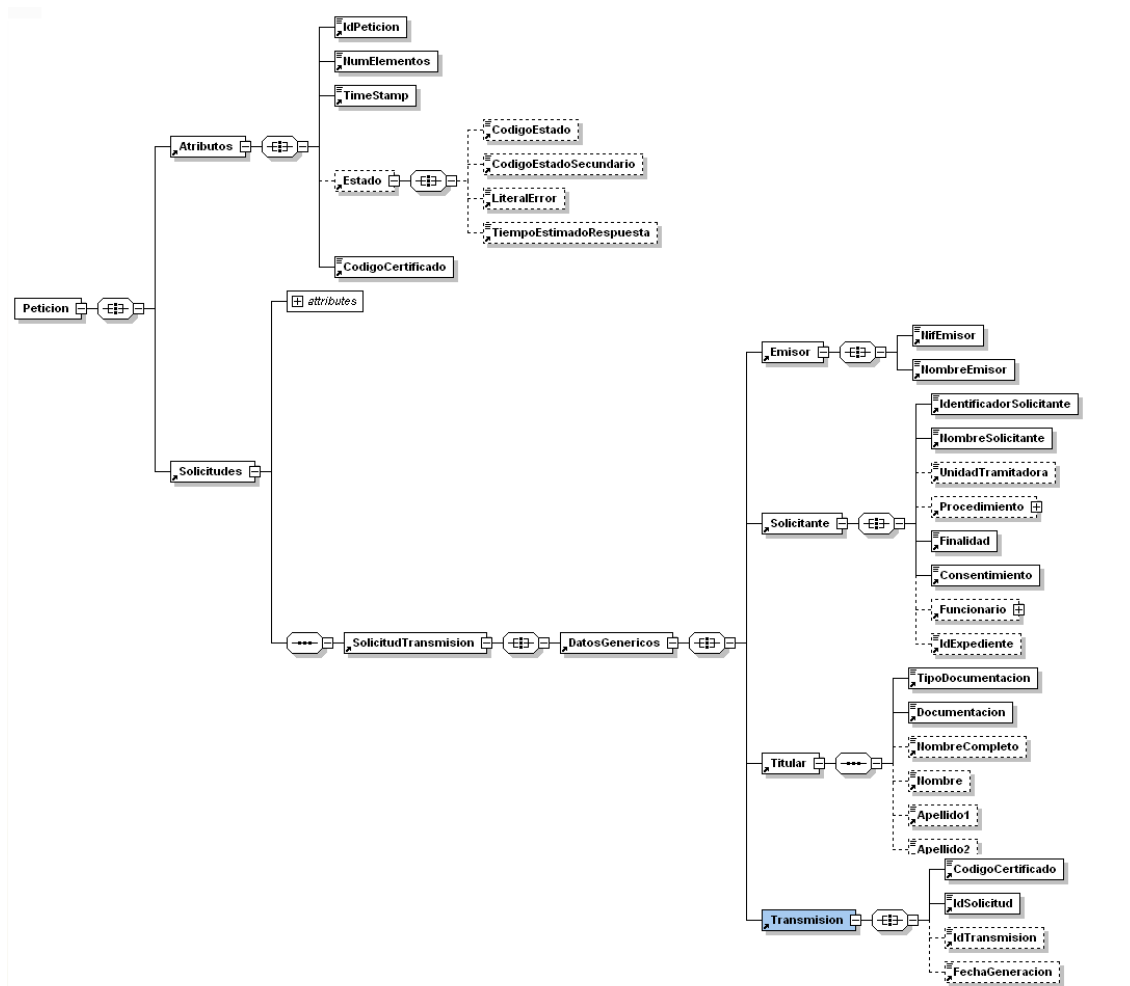
## 7.3 Emisor que simula el servicio Q2827003ATGSS001

Dentro de la distribución se proporciona un emisor de prueba que simula el funcionamiento un emisor real de la TGSS concretamente el del servicio Q2827003ATGSS001. En las siguientes líneas se hablará de sus características, como programarlo y como poder realizar las pruebas, distinguiremos entre el funcionamiento síncrono y asíncrono.

### 7.3.1 Funcionamiento Síncrono

#### 7.3.1.1 Características

Es un emisor que responde a **peticiones síncronas** con el siguiente esquema:

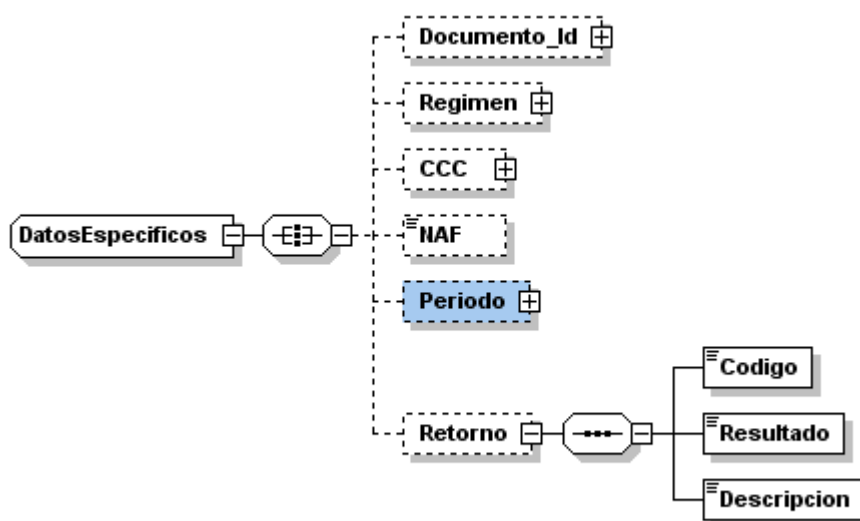


Este servicio **es un servicio de esquema V2**, que dependiendo del nif que viaje en la petición responderá:

- **Código S** → El titular está al corriente de todas sus obligaciones contraídas con la Seguridad Social: "El titular está al corriente de las obligaciones".
- **Código N** → El usuario **NO** está al corriente de todas las obligaciones contraídas con la Seguridad Social: "El titular no está al corriente de las obligaciones".

- **Código 0251** → El tipo de la documentación que viaja en la petición no es correcta: "*Tipo de documentación incorrecta*"
- **Código 0252** → La documentación que viaja en la petición es incorrecta: "*Documentación Incorrecta*"
- **Código 0253** → El titular que se está consultando tiene más de un identificador: "*La persona física o jurídica tiene más de un identificador principal*"
- **Código 0254** → El titular que se está consultando no está dado de alta dentro de la Seguridad Social como empresario: "*La persona física o jurídica no figura inscrita como empresario en el sistema de la Seguridad Social.*"
- **Código 0255** → El titular que se está consultado le falta algún tipo de dato: "*La persona física o jurídica figura inscrita como empresario en el sistema de la Seguridad Social pero no tiene asignado un CCC principal en ningún Régimen del sistema de la Seguridad Social.*"
- **Código 0256** → Error genérico del servidor: "*Respuesta no reconocida del Host.*"

Sólo uno de estos códigos podrá aparecer en la respuesta del servicio, concretamente en el nodo de Datos Específicos que tiene la siguiente estructura:



La descripción de estos campos sería la siguiente:

Nodo	Descripción
<b>Retorno/Codigo</b>	Código de respuesta del servicio puede tomar los valores 0, 0251, 0252, 0253, 0254, 0255 y 0256
<b>Retorno/Resultado</b>	Este campo puede tomar los siguientes tres valores: S, N, E
<b>Retorno/Descripcion</b>	Descripción asociado al código que devuelva el servicio.

Un ejemplo de respuesta podría ser la siguiente:

```
<Respuesta xmlns="http://www.map.es/scsp/esquemas/V2/respuesta">
```

```
<Atributos>
  <IdPeticion>TESTBRK30000067C</IdPeticion>
  <NumElementos>1</NumElementos>
  <TimeStamp>2011-03-31T16:31:55.541+02:00</TimeStamp>
  <Estado>
    <CodigoEstado>0003</CodigoEstado>
    <CodigoEstadoSecundario/>
    <LiteralError>Peticion procesada correctamente.</LiteralError>
    <TiempoEstimadoRespuesta>0</TiempoEstimadoRespuesta>
  </Estado>
  <CodigoCertificado>Q2827003ATGSS001</CodigoCertificado>
</Atributos>
<Transmisiones>
  <TransmisionDatos>
    <DatosGenericos>
      <Emisor>
        <NifEmisor>Q2827003A</NifEmisor>
        <NombreEmisor>TGSS</NombreEmisor>
      </Emisor>
      <Solicitante>
        <IdentificadorSolicitante>S2811001C</IdentificadorSolicitante>
        <NombreSolicitante>MPR</NombreSolicitante>
        <Finalidad>Prueba de Estar al corriente de Pago
          con la Seguridad Social (Q2827003ATGSS001)</Finalidad>
        <Consentimiento>Si</Consentimiento>
        <Funcionario>
          <NombreCompletoFuncionario>ANGEL
            JAVIER SALIDO MARTINEZ</NombreCompletoFuncionario>
          <NifFuncionario>47046991P</NifFuncionario>
        </Funcionario>
      </Solicitante>
      <Titular>
        <TipoDocumentacion>CIF</TipoDocumentacion>
        <Documentacion>A78518966</Documentacion>
        <NombreCompleto>Juan Prueba
          Prueba</NombreCompleto>
        </Titular>
      </Transmision>
    </DatosGenericos>
    <DatosEspecificos>
      <CodigoCertificado>Q2827003ATGSS001</CodigoCertificado>
      <IdSolicitud>TESTBRK30000067C</IdSolicitud>
      <IdTransmision>TESTBRK30000067C</IdTransmision>
      <FechaGeneracion>2011-03-
        31T16:31:55.541+02:00</FechaGeneracion>
    </DatosEspecificos>
  </Transmision>
</Transmisiones>
  <Retorno>
    <Resultado>n</Resultado>
    <Codigo>0</Codigo>
    <Descripcion>El titular no está al corriente de las
      obligaciones</Descripcion>
  </Retorno>
</DatosEspecificos>
</TransmisionDatos>
</Transmisiones>
</Respuesta>
```

**\*\*Nota:** Para más información ver el documento de integración Servicio de Verificación de Estar al Corriente de Pagos para Becas y Subvenciones que se encuentra en <http://administracionelectronica.gob.es/es/ctt/svd>

### 7.3.1.2 Programación y simulación del emisor Q2827003ATGSS001 versión síncrona

Para desarrollar y simular su comportamiento nos basaremos en un fichero de propiedades, denominado *scsp-tgss-cp-demo.properties*, que tiene la siguiente estructura:

```
nif.XXXXXXXXXX=true
nif.ZZZZZZZZ=false
nif.YYYYYYYY=true
nif.77777777=true
nif.88888888=false
nif.99999999=false

msg.e.s=El titular está al corriente de las obligaciones
msg.e.n=El titular no está al corriente de las obligaciones
msg.e.0251=Tipo de documentación incorrecta.
msg.e.0252=Documentación Incorrecta
msg.e.0253=La persona física o jurídica tiene más de un identificador principal.
msg.e.0254=La persona física o jurídica no figura inscrita como empresario en el sistema de la Seguridad Social.
msg.e.0255=La persona física o jurídica figura inscrita como empresario en el sistema de la Seguridad Social pero no tiene asignado un CCC principal en ningún Régimen del sistema de la Seguridad Social.
msg.e.0256=Respuesta no reconocida del Host.
```

En este archivo podemos ver dos patrones de propiedades diferentes:

- nif.[**número de la documentación**]-> Estas propiedades tienen asociados dos tipos de valores true/false que nos indica si el nif de la persona sobre la que se está consultando, está o no al corriente de las obligaciones con la Seguridad Social, entonces si la propiedad tiene valor true, el usuario estará al corriente, si no el usuario no estará al corriente con las obligaciones. Puede haber tantas propiedades con este formato como se quiera, un ejemplo sería el siguiente:

```
nif.XXXXXXXXXX=true
nif.YYYYYYYY=false
```

El valor que se encuentra entre corchetes ha de sustituirse por el número de la documentación que se le quiere asociar si está al corriente de pago o no.

- msg.e.[**código**] -> Sólo puede haber 8 propiedades con este formato ya que el servicio de la AEAT103I sólo devuelve cuatro tipos de código S, N, 0251, 0252, 0253, 0254, 0255 y 0256, que tienen asociada su descripción correspondiente, entonces los cuatro propiedades que pueden aparecer son los siguientes:

```
msg.e.s=El titular está al corriente de las obligaciones
msg.e.n=El titular no está al corriente de las obligaciones
msg.e.0251=Tipo de documentación incorrecta.
msg.e.0252=Documentación Incorrecta
msg.e.0253=La persona física o jurídica tiene más de un identificador principal.
msg.e.0254=La persona física o jurídica no figura inscrita como empresario en el sistema de la Seguridad Social.
msg.e.0255=La persona física o jurídica figura inscrita como empresario en el sistema de la Seguridad Social pero no tiene asignado un CCC principal en ningún Régimen del sistema de la Seguridad Social.
msg.e.0256=Respuesta no reconocida del Host
```

En caso de que se añadiera un nuevo código simplemente hay que añadir una nueva propiedad con el formato indicado anteriormente.

Una vez que se tiene definido el fichero de propiedades que se ubicará en la carpeta source para que tenga acceso desde el classpath del emisor, se debe crear el backoffice de prueba siguiendo los pasos indicados "[en el punto 6 Creación del BackOffice](#)"

La diferencia con los pasos indicados en la creación del backoffice radica en la forma de generar los datos específicos, que seguirá siguiendo un documento XML pero la forma de generar sus nodos se deberá hacer teniendo en cuenta el esquema de datos específicos ([véase el punto](#)



7.3.1.1 *Características*), un posible código que genera los datos específicos según el esquema sería el siguiente:

```
private int estaAlcorriente(String nif)throws Exception{
    boolean estaAlcorriente = false;
    String valorPropietie= properties.getProperty("nif."+nif);
    if(valorPropietie != null){
        estaAlcorriente = Boolean.parseBoolean(properties.getProperty("nif."+nif));
        if(estaAlcorriente){
            return 1;
        }else{
            return 2;
        }
    }else{
        return 0;
    }
}

private String descripcionAlcorriente(String respuesta){
    String descripcion =properties.getProperty("msg.e."+respuesta);

    return descripcion+"-"+respuesta;
}

public static boolean isNifNie(String nif){
    log.debug("NIF "+nif);
    //si es NIE, eliminar la x,y,z inicial para tratarlo como nif
    if
(nif.toUpperCase().startsWith("X")||nif.toUpperCase().startsWith("Y")||nif.toUpperCase().startsWith("Z")){
        nif = nif.substring(1);
    }
    Pattern nifPattern =
    Pattern.compile("(\\d{1,8})([TRWAGMYFPDXBNJZSQVHLCKEtrwagmyfpdxbnjzsqvhlcke])");
    Matcher m = nifPattern.matcher(nif);
    if(m.matches()){
        String letra = m.group(2);
        //Extraer letra del NIF
        String letras = "TRWAGMYFPDXBNJZSQVHLCKE";
        int dni = Integer.parseInt(m.group(1));
        dni = dni % 23;
        String reference = letras.substring(dni,dni+1);

        if (reference.equalsIgnoreCase(letra)){
            log.debug("son iguales. Es NIF. "+letra+" "+reference);
            return true;
        }else{
            log.debug("NO son iguales. NO es NIF. "+letra+" "+reference);
            return false;
        }
    }else{
        return false;
    }
}

public boolean validaCif(String cif){
    Pattern p2 = Pattern.compile("^([a-zA-Z]{1}[0-9]{8})$");
    Matcher m2 = p2.matcher(cif);
    return m2.matches();
}

private int getNumeroAleatorio(int semilla){
    Random rand = new Random();
    int numMsg = rand.nextInt(semilla)+1;
    return numMsg;
}

private Element generarDatosEspecificos(String descripcion) throws ParserConfigurationException{
```

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);

String[] descripcionCodigo = descripcion.split("-");

Document tmp = factory.newDocumentBuilder().newDocument();

Element datosEspecificos = tmp.createElement("DatosEspecificos");
Element retorno = tmp.createElement("Retorno");

Element resultado = tmp.createElement("Resultado");

Element codigoNodo = tmp.createElement("Codigo");

if(S.equalsIgnoreCase(descripcionCodigo[1])
    ||N.equalsIgnoreCase(descripcionCodigo[1])){
    codigoNodo.appendChild(tmp.createTextNode(CODIGO_CERO));
    resultado.appendChild(tmp.createTextNode(descripcionCodigo[1]));
}else{
    resultado.appendChild(tmp.createTextNode(E));
    codigoNodo.appendChild(tmp.createTextNode(descripcionCodigo[1]));
}

Element descripcionNodo = tmp.createElement("Descripcion");
descripcionNodo.appendChild(tmp.createTextNode(descripcionCodigo[0]));

retorno.appendChild(codigoNodo);
retorno.appendChild(resultado);
retorno.appendChild(descripcionNodo);
datosEspecificos.appendChild(retorno);

datosEspecificos.setAttribute("xmlns", XMLNS);

tmp.appendChild(datosEspecificos);

try{
    TransformerFactory tranFactory = TransformerFactory.newInstance();
    Transformer aTransformer = tranFactory.newTransformer();

    Source src = new DOMSource(tmp);
    Result dest = new StreamResult(System.out);
    aTransformer.transform(src, dest);
}catch(Exception ex){
    ex.printStackTrace();
}

return tmp.getDocumentElement();
}

private Respuesta ejecutarOperacion(Peticion pet) throws ScspException{
    Respuesta respuesta = new Respuesta();
    try{
        properties.load(Aeat103IBackOffice.class.getResourceAsStream("/scsp-tgss-cp-demo.properties"));

        ArrayList<TransmissionDatos> listaTransmissionDatos = new ArrayList<TransmissionDatos>();
        for (int i = 0; i < pet.getSolicitudes().getSolicitudTransmission().size(); i++) {
            SolicitudTransmission solicitudTransmission =
pet.getSolicitudes().getSolicitudTransmission().get(i);
            DatosGenericos datosGenericosPeticion = solicitudTransmission.getDatosGenericos();

            Solicitante datosSolicitantes = datosGenericosPeticion.getSolicitante();
            //Aqui se muestran los datos del solicitante de la peticion recibida.
            log.info("Datos del solicitante recibidos:");
            log.info(" - Identificador solicitante --> " + datosSolicitantes.getIdentificadorSolicitante());
            log.info(" - Nombre solicitante --> " + datosSolicitantes.getNombreSolicitante());
            log.info(" - Finalidad de la comunicacion --> " + datosSolicitantes.getFinalidad());

            Titular titular = datosGenericosPeticion.getTitular();

            log.info("Datos del titular recibidos:");
            log.info(" - Nombre completo del titular --> " + titular.getNombreCompleto());
```

```
log.info(" - Tipo documentacion del titular --> " + titular.getTipoDocumentacion().name());
log.info(" - Documentacion del titular --> " + titular.getDocumentacion());
log.info("El servicio de la TGSS001 no posee datos específicos");

log.info("##### COMIENZO Generando Respuesta");
respuesta.setAtributos(pet.getAtributos());
respuesta.getAtributos().setEstado(new Estado());
respuesta.getAtributos().getEstado().setCodigoEstado("0003");
respuesta.getAtributos().getEstado().setCodigoEstadoSecundario("");
respuesta.getAtributos().getEstado().setLiteralError("Petición procesada correctamente.");
respuesta.getAtributos().getEstado().setTiempoEstimadoRespuesta(0);

log.info("##### COMIENZO Generando datos Especificos de Respuesta");
int estaAlcorriente = estaAlcorriente(titular.getDocumentacion());
String descripcion = "";
if(estaAlcorriente == 1){
    descripcion = descripcionAlcorriente("s");
}else{
    if(estaAlcorriente == 2){
        descripcion = descripcionAlcorriente("n");
    }else{
        if(!esTipoDocumentacionCorrecta(titular.getTipoDocumentacion().name())){
            descripcion = descripcionAlcorriente("0251");
        }else{
            String codigoRespuesta = "";

            if(titular.getTipoDocumentacion().name().equalsIgnoreCase(TipoDocumentacion.NIE.name())
            ||titular.getTipoDocumentacion().name().equalsIgnoreCase(TipoDocumentacion.NIF.name())
            ||titular.getTipoDocumentacion().name().equalsIgnoreCase(TipoDocumentacion.CIF.name())){

                if(titular.getTipoDocumentacion().name().equalsIgnoreCase(TipoDocumentacion.NIF.name())){

                    if(!isNifNie(titular.getDocumentacion())){
                        codigoRespuesta="0252";
                    }

                    if(!validaCif(titular.getDocumentacion())){
                        codigoRespuesta="0252";
                    }
                }
                if("").equalsIgnoreCase(codigoRespuesta)){
                    codigoRespuesta =
"025"+(getNumeroAleatorio(3) + 3);
                }
            }else{
                codigoRespuesta = "025"+(getNumeroAleatorio(3)
+ 3);
            }
            descripcion =descripcionAlcorriente(codigoRespuesta);
        }
    }
}

log.info("- El titular esta --> "+descripcion);

TransmisionDatos datos = new TransmisionDatos();

datos.setDatosGenericos(datosGenericosPetición);
datos.setDatosEspecificos(generarDatosEspecificos(descripcion));

log.info("##### FIN Generando datos Especificos de Respuesta");

listaTransmisionDatos.add(datos);
}
Transmisiones transmisiones = new Transmisiones();
```

```
transmisiones.setTransmisionDatos(listaTransmisionDatos);

respuesta.setTransmisiones(transmisiones);
log.info("##### FIN Generando Respuesta");
}catch(Exception ex){

String []a={""}
throw ScspException.getScspException(ScspExceptionConstants.ErrorBackOfficePruebas, "Error al procesar la
notificacion sincrona.");

}
return respuesta;
}
```

El funcionamiento de estos métodos es el siguiente:

- **int** estaAlcorriente(String nif) → Este método se encarga de ir al archivo de propiedades y dependiendo del nif que se le pase por parámetro devolverá un valor 0 , 1 o 2, para ello lo que hace es concatenar la cadena "nif." con el número de nif que se le pasa por parámetro y con ese valor de propiedad se coge el valor correspondiente, en caso de ser true el método devolverá un 1, en caso de false devolverá un 2 y en caso de meter un nif inexistente, es decir, que concatenando "nif" más el número de nif pasado por parámetro no existiese esa propiedad se devolverá valor 0.
- String descripcionAlcorriente(String respuesta) → Este método se encarga de devolver la descripción del mensaje a devolver descripción de la respuesta que se vaya a enviar.
- **boolean** isNifNie(String nif) → Este método valida que la documentación que le llega es un NIF o NIE correcto.
- **boolean** validaCif(String cif) → Este método se encarga de comprobar que el CIF que se le pasa por parámetro es correcto.
- **int** getNumeroAleatorio(int semilla) → Este método se encarga de obtener un número aleatorio según una semilla.
- **Element generateDatosEspecificos(String descripcion)** → Este método se encarga de generar un documento XML con la estructura acorde con los datos específicos, entonces a este método se le pasa una cadena que tiene el siguiente formato "descripcion-codigo" entonces lo que se hace es separa la descripción y el código haciendo un split por la cadena "-" que nos devuelve un array de dos posiciones donde la primera posición (0) viene la descripción y en la segunda posición (1) con el código de respuesta, entonces cada valor se añade al nodo correspondiente generándose un objeto DOM que posteriormente se añade a la respuesta que se está generando, para más información sobre la composición de los datos específicos de este servicio "véase el punto 7.3.1.1 Características".

### 7.3.1.3 Como probar la simulación del servicio.

Para probar la ejecución de este servicio en los datos específicos de la petición, deberá venir una fecha que cumpla alguno de los siguientes casos:

NIF	RESPUESTA
XXXXXXXXXX	<p>El titular <b>SI</b> está al corriente de las obligaciones con la Seguridad Social.</p> <p><b>Código de respuesta</b>→ 0</p> <p><b>Resultado</b> → S</p> <p><b>Descripción de respuesta</b>→ <u>El titular está al corriente de las obligaciones.</u></p>

<b>ZZZZZZZZZ</b>	<p>El titular <b>NO</b> está al corriente de las obligaciones con la Seguridad social</p> <p><b>Código de respuesta</b> → 0</p> <p><b>Resultado</b> → N</p> <p><b>Descripción de respuesta</b> → <u>El titular no está al corriente de las obligaciones de pago.</u></p>
<b>YYYYYYYYY</b>	<p>El titular <b>SI</b> está al corriente de las obligaciones con la Seguridad Social.</p> <p><b>Código de respuesta</b> → 0</p> <p><b>Resultado</b> → S</p> <p><b>Descripción de respuesta</b> → <u>El titular está al corriente de las obligaciones.</u></p>
<b>777777777</b>	<p>El titular <b>SI</b> está al corriente de las obligaciones con la Seguridad Social.</p> <p><b>Código de respuesta</b> → 0</p> <p><b>Resultado</b> → S</p> <p><b>Descripción de respuesta</b> → <u>El titular está al corriente de las obligaciones.</u></p>
<b>888888888</b>	<p>El titular <b>NO</b> está al corriente de las obligaciones con la Seguridad social</p> <p><b>Código de respuesta</b> → 0</p> <p><b>Resultado</b> → N</p> <p><b>Descripción de respuesta</b> → <u>El titular no está al corriente de las obligaciones de pago.</u></p>
<b>999999999</b>	<p>El titular <b>NO</b> está al corriente de las obligaciones con la Seguridad social</p> <p><b>Código de respuesta</b> → 0</p> <p><b>Resultado</b> → N</p> <p><b>Descripción de respuesta</b> → <u>El titular no está al corriente de las obligaciones de pago.</u></p>
<b>Otro NIF, NIE o CIF</b>	<p>En caso de recibir otro NIF, NIE o CIF, se podrán obtener alguno de los siguientes códigos y descripciones:</p> <p><b>Código de respuesta</b> → 0251, 0252, 0253, 0254, 0255 o 0256</p> <p><b>Resultado</b> → E</p> <p><b>Descripción de la respuesta :</b></p> <p><b>0251</b> → Tipo de documentación incorrecta</p>

**0252** → Documentacion Incorrecta

**0253** → La persona física o jurídica tiene más de un identificador principal

**0254** → La persona física o jurídica no figura inscrita como empresario en el sistema de la Seguridad Social.

**0255** → La persona física o jurídica figura inscrita como empresario en el sistema de la Seguridad Social pero no tiene asignado un CCC principal en ningún Régimen del sistema de la Seguridad Social.

**0256** → Respuesta no reconocida del Host

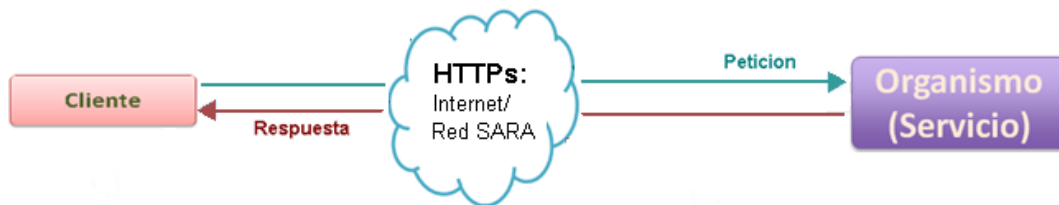
## 7.3.2 Funcionamiento asíncrono

### 7.3.2.1 Características

La diferencia desde el punto de vista de la programación entre el funcionamiento síncrono y asíncrono radica en la secuenciación de operaciones, para entender esta diferencia debemos ver los dos siguientes gráficos:

- Comunicación síncrona

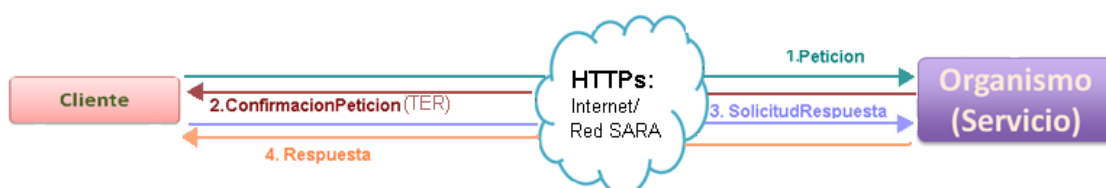
En este caso el organismo, emisor o servicio responde inmediatamente.



- Comunicación asíncrona

En este caso las comunicaciones son más complejas:

- El cliente envía una petición con los datos concretos que solicita al servicio.
- El servicio envía un mensaje confirmando la recepción de la solicitud, pero sin enviar el resultado del servicio. En cambio le envía una “fecha TER” (tiempo estimado de respuesta) a partir de la cual el cliente podrá solicitar la respuesta definitiva del servicio.
- Una vez cumplido el TER, el cliente envía un mensaje solicitando una respuesta definitiva al servicio que solicitó en el pasado.
- El servicio entonces le envía la respuesta definitiva.



### 7.3.2.2 Programación y simulación del emisor Q2827003ATGSS001 versión síncrona.

Para el desarrollo del comportamiento asíncrono de este servicio se deben desarrollar dos métodos:

- Notificar Asíncrono
- Solicitud de Respuesta

Para desarrollar estos puntos “véase punto 6 Interface BackOffice”, la única diferencia con lo expuesto en este punto, es el desarrollo de los datos específicos que se deben generar en la respuesta a la solicitud de respuesta que se realiza para obtener la respuesta definitiva de una petición asíncrona, para ello “véase el punto 7.3.1.2 Programación y simulación del emisor Q2827003ATGSS001 versión síncrona”, la respuesta definitiva a una petición asíncrona se genera de la misma forma que una petición síncrona.

Entonces teniendo en cuenta estas premisas tendremos los siguientes métodos:

- Notificación de peticiones asíncronas:

```
public ConfirmacionPetición NotificarAsíncrono(Petición pet)
    throws ScspException {
    log.info("##### COMIENZO EMISOR PRUEBA SINCRONO TGSS001
#####");
    log.info("Procesando petición asíncrona por " + getClass() + ".");
    try {
        int tiempoRespuesta = 5;
        ConfirmacionPetición respuesta = new ConfirmacionPetición();

        log.info("##### Generando Respuesta...");

        respuesta.setAtributos(pet.getAtributos());
        respuesta.getAtributos().setEstado(new Estado());
        respuesta.getAtributos().getEstado().setCodigoEstado(CodigoEstado.EnProceso);
        respuesta.getAtributos().getEstado().setCodigoEstadoSecundario("");
        respuesta.getAtributos().getEstado().setLiteralError("Petición procesada correctamente.");
        respuesta.getAtributos().getEstado().setTiempoEstimadoRespuesta(tiempoRespuesta);
        respuesta.getAtributos().setTimeStamp(pet.getAtributos().getTimeStamp());

        log.info("##### Respuesta Generada....");
        log.info("##### FIN EMISOR PRUEBA SINCRONO TGSS001
#####");
        return respuesta;
    } catch (Exception e) {
        String []a={""}
    }
    throw ScspException.getScspException(ScspExceptionConstants.ErrorBackOfficePruebas, "Error al procesar la
notificación síncrona.");
}
```

- Notificación de solicitudes de respuesta:

```
log.info("##### COMIENZO EMISOR SOLICITUD RESPUESTA TGSS001 #####");
log.info("Procesando solicitud de respuesta por " + getClass() + ".");

log.info("Recuperando petición de la base de datos....");

Respuesta respuesta = new Respuesta();

TokenDao tokenDao =(TokenDao)StaticContextSupport.getContextInstance().getBean("tokenDao");

Token token = tokenDao.select(sr.getAtributos().getIdPetición()+"_Petición");
PeticiónFactory peticiónFactory= null;
```



```

try{
    peticiónFactory = new PeticiónFactory();
}catch(XPathExpressionException ex){
    String []a={""}
throw ScspException.getScspException(ScspExceptionConstants.ErrorBackOfficePruebas, "Error al procesar la
notificación sincrónica.");
}

if(token == null){
String []a={""}
throw ScspException.getScspException("0241", " La petición no existe en el sistema.");

}else{
    Document tokenXML = Utils.string2DOM(token.getDatos());
    log.info(Utils.DOM2String(tokenXML));
    Petición peticiónBBDD = peticiónFactory.getPetición(tokenXML.getDocumentElement());
    respuesta = ejecutarOperación(peticiónBBDD);
    log.info("##### FIN EMISOR PRUEBA SOLICITUD RESPUESTA
TGSS001 #####");
    return respuesta;
}
}

```

De este método se debe destacar que para simular el comportamiento asíncrono se siguen los siguientes pasos:

1. Obtener token (Petición en formato XML) de la petición de la base de datos a través del IdPetición solicitado.
2. Serializar el documento XML obtenido a objetos.
3. Una vez se tiene el objeto, ejecutar el mismo código que si fuese una petición síncrona ya que el comportamiento es análogo "véase el punto 7.3.1.2 Programación y simulación del emisor Q2827003ATGSS001 versión síncrona"
4. Devolver la respuesta obtenida.

### 7.3.2.3 Como probar la simulación del servicio

Para simular el comportamiento asíncrono, se deben seguir estos pasos:

1. Generar petición Asíncrona, un ejemplo podría ser el siguiente:

```

<Petición xmlns="http://www.map.es/scsp/esquemas/V2/petición">
  <Atributos>
    <IdPetición>TESTBRK300000688</IdPetición>
    <NumElementos>1</NumElementos>
    <TimeStamp>2011-03-31T18:54:09.432+02:00</TimeStamp>
    <CodigoCertificado>Q2827003ATGSS001</CodigoCertificado>
  </Atributos>
  <Solicitudes>
    <SolicitudTransmisión>
      <DatosGenericos>
        <Emisor>
          <NifEmisor>Q2827003A</NifEmisor>
          <NombreEmisor>TGSS</NombreEmisor>
        </Emisor>
        <Solicitante>
          <IdentificadorSolicitante>S2811001C</IdentificadorSolicitante>
          <NombreSolicitante>MPR</NombreSolicitante>
          <Finalidad>Prueba de Estar al corriente de Pago
con la Seguridad Social (Q2827003ATGSS001)</Finalidad>
          <Consentimiento>Si</Consentimiento>
          <Funcionario>
            <NombreCompletoFuncionario>ANGEL
JAVIER SALIDO MARTINEZ</NombreCompletoFuncionario>

```



```

<NifFuncionario>47046991P</NifFuncionario>
</Funcionario>
</Solicitante>
<Titular>
  <TipoDocumentacion>NIF</TipoDocumentacion>
  <Documentacion>47046991</Documentacion>
  <NombreCompleto>Juan Prueba
Prueba</NombreCompleto>
</Titular>
<Transmision>
  <CodigoCertificado>Q2827003ATGSS001</CodigoCertificado>
  <IdSolicitud>TESTBRK300000688</IdSolicitud>
  <IdTransmision>TESTBRK300000688</IdTransmision>
  <FechaGeneracion>2011-03-
31T18:54:09.432+02:00</FechaGeneracion>
</Transmision>
</DatosGenericos>
</SolicitudTransmision>
</Solicitudes>
</Peticion>

```

2. Enviar dicha petición asíncrona.
3. Quedarnos con el Id de petición  
En este caso TESTBRK300000688
4. Realizar una solicitud de respuesta con dicho id de petición.

```

<?xml version="1.0" encoding="UTF-8"?>
<SolicitudRespuesta xmlns="http://intermediacion.redsara.es/scsp/esquemas/ws/solicitudRespuesta">
  <Atributos>
    <CodigoCertificado>Q2827003ATGSS001</CodigoCertificado>
    <IdPetición> TESTBRK300000688</IdPetición>
    <NumElementos>1</NumElementos>
  </Atributos>
</SolicitudRespuesta>

```

5. Procesar la respuesta definitiva obtenida, en este punto se deben obtener los mismos resultados que los expresados en la tabla del “*punto 7.3.1.3 Como probar la simulación del servicio*”. Un ejemplo de respuesta podría ser:

```

<Respuesta xmlns="http://www.map.es/scsp/esquemas/V2/respuesta">
  <Atributos>
    <IdPetición>TESTBRK300000688</IdPetición>
    <NumElementos>1</NumElementos>
    <TimeStamp>2011-03-31T18:54:09.432+02:00</TimeStamp>
    <Estado>
      <CodigoEstado>0003</CodigoEstado>
      <CodigoEstadoSecundario/>
      <LiteralError>Petición procesada correctamente.</LiteralError>
      <TiempoEstimadoRespuesta>0</TiempoEstimadoRespuesta>
    </Estado>
    <CodigoCertificado>Q2827003ATGSS001</CodigoCertificado>
  </Atributos>
  <Transmisiones>
    <TransmisionDatos>
      <DatosGenericos>
        <Emisor>
          <NifEmisor>Q2827003A</NifEmisor>
          <NombreEmisor>TGSS</NombreEmisor>
        </Emisor>
      </DatosGenericos>
    </TransmisionDatos>
  </Transmisiones>
  <IdentificadorSolicitante>S2811001C</IdentificadorSolicitante>
  <NombreSolicitante>MPR</NombreSolicitante>
</Respuesta>

```

```
con la Seguridad Social (Q2827003ATGSS001)</Finalidad>
<Finalidad>Prueba de Estar al corriente de Pago
<Consentimiento>Si</Consentimiento>
<Funcionario>
  <NombreCompletoFuncionario>ANGEL
JAVIER SALIDO MARTINEZ</NombreCompletoFuncionario>
  <NifFuncionario>47046991P</NifFuncionario>
    </Funcionario>
    </Solicitante>
    <Titular>
      <TipoDocumentacion>NIF</TipoDocumentacion>
      <Documentacion>47046991</Documentacion>
      <NombreCompleto>Juan Prueba
Prueba</NombreCompleto>
    </Titular>
    <Transmision>
      <CodigoCertificado>Q2827003ATGSS001</CodigoCertificado>
      <IdSolicitud>TESTBRK300000688</IdSolicitud>
      <IdTransmision>TESTBRK300000688</IdTransmision>
      <FechaGeneracion>2011-03-
31T18:54:09.432+02:00</FechaGeneracion>
    </Transmision>
    </DatosGenericos>
    <DatosEspecificos
xmlns="http://intermediacion.redsara.es/scsp/esquemas/datosespecificos">
      <Retorno>
        <Resultado>E</Resultado>
        <Codigo>0252</Codigo>
        <Descripcion>Documentacion
Incorrecta</Descripcion>
      </Retorno>
    </DatosEspecificos>
    </TransmisionDatos>
  </Transmisiones>
</Respuesta>
```

## 7.4 Emisor con Backoffice de Pruebas

### 7.4.1 Funcionamiento Asíncrono

Para el desarrollo del comportamiento asíncrono de este servicio se deben desarrollar dos métodos:

- Notificar Asíncrono
- Solicitud de Respuesta

Este backoffice tiene como misión enseñar la forma de parsear un String con formato XML y obtener datos de alguno de los nodos de este documento.

Para desarrollar estos puntos “véase *punto 6 Interface BackOffice*”, la única diferencia con lo expuesto en este punto, es que se obtiene una cadena de caracteres con formato XML, se parsea y se obtienen datos de dicho documento XML.

Entonces teniendo en cuenta estas premisas tendremos los siguientes métodos:

#### 7.4.1.1 Programación y simulación del emisor que parsea XML versión asíncrona

- Notificación de peticiones asíncronas:

```
public ConfirmacionPetition NotificarAsincrono(Petition pet)
    throws ScspException {

    log.info("Procesando petition asincrona por " + getClass() + ".");
    try {
        int tiempoRespuesta = 5;
        ConfirmacionPetition respuesta = new ConfirmacionPetition();

        log.info("##### Generando Respuesta...");

        respuesta.setAtributos(pet.getAtributos());
        respuesta.getAtributos().setEstado(new Estado());
        respuesta.getAtributos().getEstado().setCodigoEstado(CodigoEstado_EnProceso);
        respuesta.getAtributos().getEstado().setCodigoEstadoSecundario("");
        respuesta.getAtributos().getEstado().setLiteralError("Petition procesada correctamente.");
        respuesta.getAtributos().getEstado().setTiempoEstimadoRespuesta(tiempoRespuesta);
        respuesta.getAtributos().setTimeStamp(pet.getAtributos().getTimeStamp());

        log.info("##### Respuesta Generada....");

        return respuesta;
    } catch (Exception e) {

        String []a={""}
        throw ScspException.getScspException(ScspExceptionConstants.ErrorBackOfficePruebas, "Error durante la notificacion asincrona.");

    }
}
```

- Notificación de solicitudes de respuesta:

```
public Respuesta SolicitudRespuesta(
    es.scsp.bean.common.SolicitudRespuesta sr) throws ScspException {

    printDemo();

    Atributos atributos = sr.getAtributos();
    /*
```

```
* Con el metodo generateRespuesta de la clase RespuestaGenerator se
* rellenan con datos de prueba los datos genéricos de la respuesta
* que se está generando como respuesta a la solicitud realizada.
*/

log.info("##### COMIENZO EMISOR SOLICITUD RESPUESTA
DEMOBACKOFFICE #####");
log.info("Procesando solicitud de respuesta por " + getClass() + ".");

log.info("Recuperando peticion de la base de datos....");

TokenDao tokenDao =(TokenDao)StaticContextSupport.getContextInstance().getBean("tokenDao");

Token token = tokenDao.select(sr.getAtributos().getIdPeticion()+"_Peticion");

mostrarDatos(token.getDatos());

log.info("##### Generando datos genericos de la respuesta.");
Respuesta respuesta = new RespuestaGenerator().generateRespuesta(atributos);
/*
* Se generan los datos especificos de prueba.
*/
log.info("##### Generando los datos especificos de la respuesta.");
Object de = generarDatosEspecificosRespuesta(sr.getAtributos().getCodigoCertificado());
for(int i=0;i < Integer.parseInt(atributos.getNumElementos()); i++) {
    respuesta.getTransmisiones().getTransmisionDatos().get(i).setDatosEspecificos(de);
}
log.info("##### Respuesta Generada.");
return respuesta;
}
```

De este método se debe destacar que para simular el comportamiento asíncrono se siguen los siguientes pasos:

1. Obtener token (Petición en formato XML) de la petición de la base de datos a través del IdPetición solicitado.

```
TokenDao tokenDao =(TokenDao)StaticContextSupport.getContextInstance().getBean("tokenDao");
Token token = tokenDao.select(sr.getAtributos().getIdPeticion()+"_Peticion");
```

2. Una vez se tiene, ese token que es un String, se ha de parsear para transformarlo en un documento XML, para ello se utilizarán las siguientes funciones:

```
private String fechaDeHoy() {

    Date date = new Date();
    String formato = new String("dd/MM/yyyy HH:mm:ss:SSS");
    SimpleDateFormat formatoSimple = new SimpleDateFormat(formato);
    return formatoSimple.format(date);
}

private String getIdentificadorSolicitante(Element xml){
    NodeList listaHijos =xml.getElementsByTagName("IdentificadorSolicitante");
    return listaHijos.item(0).getChildNodes().item(0).getNodeValue();
}

private String getNombreSolicitante(Element xml){
    NodeList listaHijos =xml.getElementsByTagName("NombreSolicitante");
    return listaHijos.item(0).getChildNodes().item(0).getNodeValue();
}

private String getFinalidadSolicitante(Element xml){
    NodeList listaHijos =xml.getElementsByTagName("Finalidad");
    return listaHijos.item(0).getChildNodes().item(0).getNodeValue();
}
```

```

    }

    private Document str_to_xml(String str){

        try {
            DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            InputSource is = new InputSource();
            is.setCharacterStream(new StringReader(str));

            Document doc = db.parse(is);
            return doc;
        }catch (Exception e) {
            e.printStackTrace();
        }

        return null;
    }

    private void mostrarDatos(String str){
        Document doc = str_to_xml(str);
        log.info("##### DATOS DEL XML #####");
        log.info("##### IDENTIFICADOR SOLICITANTE -->
"+getIdentificadorSolicitante(doc.getDocumentElement());
        log.info("##### NOMBRE SOLICITANTE --> "+getNombreSolicitante(doc.getDocumentElement());
        log.info("##### FINALIDAD SOLICITANTE --
>"+getFinalidadSolicitante(doc.getDocumentElement());
    }

```

Estas funciones tienen las siguientes funciones:

Nombre Función	Parámetros	Descripción
<b>fechaDeHoy</b>	<u>Entrada:</u> N/A <u>Salida:</u> Devuelve una fecha en formato String	Esta función tiene como función devolver la fecha de hoy en un tipo String, con el siguiente formato: "dd/MM/yyyy HH:mm:ss:SS"
<b>getIdentificadorSolicitante</b>	<u>Entrada:</u> Un Element de DOM <u>Salida:</u> Un objeto de tipo String	Es una función que devuelve el valor que contiene el nodo IdentificadorSolicitante
<b>getNombreSolicitante</b>	<u>Entrada:</u> Un Element de DOM <u>Salida:</u> Un objeto de tipo String	Es una función que devuelve el valor que contiene el nodo NombreSolicitante
<b>getFinalidadSolicitante</b>	<u>Entrada:</u> Un Element de DOM <u>Salida:</u> Un objeto de tipo String	Es una función que devuelve el valor que contiene el nodo Finalidad
<b>str_to_xml</b>	<u>Entrada:</u> Un String con formato XML <u>Salida:</u> Un objeto de tipo Document de DOM	Esta función parsea una cadena con formato XML y lo transforma en un objeto Document de DOM.
<b>mostrarDatos</b>	<u>Entrada:</u> Un String con formato XML	Esta función coge por parámetro una cadena de

Salida: N/A

caracteres con formato de XML, y muestra apoyándose en las funciones descritas anteriormente, los valores que contienen los nodos NombreSolicitante, Identificador solicitante y Finalidad.

- Una vez se ha realizado el parseo, se utilizará el siguiente método para realizar la generación de los datos específicos que posteriormente se introducirán dentro de cada una de las transmisiones existentes:

```
private Element generarDatosEspecificosRespuesta(String codigoCertificado) {
    DocumentBuilderFactory fac = DocumentBuilderFactory.newInstance();
    fac.setNamespaceAware(false);
    Document doc;
    try {
        doc = fac.newDocumentBuilder().newDocument();
    } catch (ParserConfigurationException e) {
        throw new RuntimeException("Error al crear el documento.", e);
    }
    Element result = doc.createElement("DatosEspecificos");
    // Generando elemento descripcion de la respuesta.
    Element descripcion = doc.createElement("RespuestaEjemplo");
    //descripcion.setTextContent("Respuesta de Backoffice V3 de prueba");
    // Generando elemento codigoCertificado de la respuesta.
    Element codigoCertificadoElement = doc.createElement("CodigoCertificado");
    if (codigoCertificado != null) {
        codigoCertificadoElement.setTextContent(codigoCertificado);
    } else {
        codigoCertificadoElement.setTextContent("No existe el codigo de certificado");
    }
    // Generando elemento fechaRespuesta de la respuesta.
    Element fechaRespuesta = doc.createElement("FechaRespuesta");
    fechaRespuesta.setTextContent(fechaDeHoy());
    result.setAttribute("xmlns", XMLNS);
    result.appendChild(descripcion);
    result.appendChild(codigoCertificadoElement);
    result.appendChild(fechaRespuesta);
    return result;
}
```

Nombre Función	Parámetros	Descripción
<b>generarDatosEspecificosRespuesta</b>	<u>Entrada:</u> String código de certificado. <u>Salida:</u> Devuelve un objeto Element de DOM.	<p>Este método se encarga de construir un XML que serán los datos específicos que irán en la respuesta, el formato del XML que se genera es el siguiente:</p> <pre>&lt;DatosEspecificos xmlns="http://intermediacion.redsara.es/scsp/esquemas/datosespecificos"&gt; &lt;descripcion&gt;Respuesta de Backoffice V3 de prueba &lt;/descripcion&gt; &lt;codigoCertificado&gt;XXXXXXXXXX&lt;/codigoCertificado&gt; &lt;fechaRespuesta&gt;XXXXXXXXXX&lt;/fechaRespuesta&gt; &lt;/DatosEspecificos&gt;</pre>

- Devolver la respuesta obtenida

### 7.4.1.2 Como probar la simulación del servicio

Para simular el comportamiento asíncrono, se deben seguir estos pasos:

1. Generar petición Asíncrona, un ejemplo podría ser el siguiente:

```
<PeticiónAsíncrona xmlns="http://intermediacion.redsara.es/scsp/esquemas/ws/petición">
  <Atributos>
    <IdPetición>3cb28d8c00005594</IdPetición>
    <NumElementos>1</NumElementos>
    <TimeStamp>2011-02-21T16:16:11.590+01:00</TimeStamp>
    <CodigoCertificado>PRU_XML</CodigoCertificado>
  </Atributos>
  <Solicitudes>
    <SolicitudTransmisión>
      <DatosGenericos>
        <Emisor>
          <NifEmisor>S2811001C</NifEmisor>
          <NombreEmisor>MPTAP</NombreEmisor>
        </Emisor>
        <Solicitante>
          <IdentificadorSolicitante>S2811001C</IdentificadorSolicitante>
          <NombreSolicitante>MPTAP</NombreSolicitante>
          <UnidadTramitadora>PRUEBA</UnidadTramitadora>
        </Solicitante>
        <Procedimiento />
        <Finalidad>Prueba de Estar al corriente de Pago con la Seguridad Social (Q2827003ATGSS001)</Finalidad>
        <Consentimiento>Si</Consentimiento>
        <Funcionario>
          <NombreCompletoFuncionario>ANGEL JAVIER SALIDO MARTINEZ</NombreCompletoFuncionario>
          <NifFuncionario>47046991P</NifFuncionario>
        </Funcionario>
        <Solicitante>
          <Titular>
            <TipoDocumentación>NIF</TipoDocumentación>
            <Documentación>XXXXXXXXXX</Documentación>
            <NombreCompleto>Juan Prueba Prueba</NombreCompleto>
          </Titular>
          <Transmisión>
            <CodigoCertificado>PRU_XML</CodigoCertificado>
            <IdSolicitud>3cb28d8c00005594</IdSolicitud>
            <IdTransmisión>3cb28d8c00005594</IdTransmisión>
            <FechaGeneración>2011-02-21T16:16:11.590+01:00</FechaGeneración>
          </Transmisión>
        </Solicitante>
      </SolicitudTransmisión>
    </Solicitudes>
  </PeticiónAsíncrona>
```

2. Enviar dicha petición asíncrona.

3. Quedarnos con el Id de petición

En este caso PREKP0000000001A

4. Realizar una solicitud de respuesta con dicho id de petición.

```
<?xml version="1.0" encoding="UTF-8"?>
<SolicitudRespuesta xmlns="http://intermediacion.redsara.es/scsp/esquemas/ws/solicitudRespuesta">
  <Atributos>
    <CodigoCertificado>PRU_XML</CodigoCertificado>
    <IdPetición>PREKP0000000001A</IdPetición>
    <NumElementos>1</NumElementos>
  </Atributos>
</SolicitudRespuesta>
```

5. Procesar la respuesta definitiva obtenida, después del parseo del XML y de mostrar por pantalla la información recogida del XML. Un ejemplo de respuesta podría ser:

```
<?xml version="1.0" encoding="UTF-8"?>
<Respuesta xmlns="http://intermediacion.redsara.es/scsp/esquemas/ws/respuesta">
  <Atributos>
    <IdPetition>PREKP0000000001A</IdPetition>
    <NumElementos>1</NumElementos>
    <TimeStamp>2011-05-04T10:15:22.773+02:00</TimeStamp>
    <Estado>
      <CodigoEstado>0003</CodigoEstado>
      <CodigoEstadoSecundario>0003</CodigoEstadoSecundario>
      <LiteralError>Petición procesada correctamente.</LiteralError>
      <TiempoEstimadoRespuesta>0</TiempoEstimadoRespuesta>
    </Estado>
    <CodigoCertificado>PRU_XML</CodigoCertificado>
  </Atributos>
  <Transmisiones>
    <TransmisionDatos>
      <DatosGenericos>
        <Emisor>
          <NifEmisor>S2833002E</NifEmisor>
          <NombreEmisor>Organismo de pruebas de Burke</NombreEmisor>
        </Emisor>
        <Solicitante>
          <IdentificadorSolicitante>idsol</IdentificadorSolicitante>
          <NombreSolicitante>nombre sol</NombreSolicitante>
          <UnidadTramitadora>unidad tramitadora</UnidadTramitadora>
          <Procedimiento>
            <CodProcedimiento>cod proc</CodProcedimiento>
            <NombreProcedimiento>nombre
proc</NombreProcedimiento>
          </Procedimiento>
          <Finalidad>finalidad</Finalidad>
          <Consentimiento>Si</Consentimiento>
          <Funcionario>
            <NombreCompletoFuncionario>nombreCompleto</NombreCompletoFuncionario>
            <NifFuncionario>NF1000111</NifFuncionario>
          </Funcionario>
          <IdExpediente>idexp</IdExpediente>
        </Solicitante>
        <Titular>
          <TipoDocumentacion>CIF</TipoDocumentacion>
          <Documentacion>C555555555</Documentacion>
          <NombreCompleto>John Doe Smith</NombreCompleto>
          <Nombre>John</Nombre>
          <Apellido1>Doe</Apellido1>
          <Apellido2>Smith</Apellido2>
        </Titular>
        <Transmision>
          <CodigoCertificado>PRU_XML</CodigoCertificado>
          <IdSolicitud>PREKP0000000001A</IdSolicitud>
          <IdTransmision>id tr</IdTransmision>
          <FechaGeneracion>fecha gen</FechaGeneracion>
        </Transmision>
      </DatosGenericos>
      <DatosEspecificos
xmlns="http://intermediacion.redsara.es/scsp/esquemas/datosespecificos">
        <RespuestaEjemplo/>
        <CodigoCertificado>PRU_XML</CodigoCertificado>
        <FechaRespuesta>04/05/2011 10:15:53:336</FechaRespuesta>
      </DatosEspecificos>
    </TransmisionDatos>
  </Transmisiones>
</Respuesta>
```

En los logs deberemos encontrar los siguientes datos:

```
##### DATOS DEL XML #####
##### IDENTIFICADOR SOLICITANTE --> S2811001C
##### NOMBRE SOLICITANTE --> MPTAP
```



##### FINALIDAD SOLICITANTE -->Prueba de Estar al corriente de Pago con la Seguridad Social (Q2827003ATGSS001)