Plataforma Autonómica de Interoperabilidad

Manual_Usuario_Generación_de_Servicios

GENERALITAT VALENCIANA
Conselleria d'Hisenda i Model Econòmic

DIRECCIÓN GENERAL DE TECNOLOGÍAS
DE LA INFORMACIÓN Y LAS COMUNICACIONES

Versión 012
Marzo de 2022

Unió Europea
Fons Europeu de Desenvolupament Regional
Una manera de fer Europa

Unión Europea
Fondo Europeo de Desarrollo Regional
Una manera de hacer Europa

1. Control del documento

1.1 Información general

Título	Manual de usuario de generación de Servicios
Creado por	DGTI
Revisado por	
Lista de distribución	
Nombre del fichero	CONSTRUCCION-Manual_Usuario_Generación_de_Servicios_v12.docx

1.2 Histórico de revisiones

Versión	Fecha	Autor	Observaciones
001	14/08/2014	DGTI	Versión inicial
002	21/08/2014	DGTI	Añadidos Anexo II y III
003	28/08/2014	DGTI	Añadido Anexo IV y revisión de contenido.
004	30/10/2015	DGTI	Añadido Anexo IV:mustunderstand y renumeración de anexos
005	25/05/2017	DGTI	Cambio en el nombre del documento. Re-estructuración del contenido, validación de versión CXF v3.1.10
006	19/06/2017	DGTI	Revisión de documento
007	20/06/2017	DGTI	Añadida nota en el apartado SoapFault. Revisión del documento.
008	09/03/2018	DGTI	Revisión documento, cambio logo.
009	26/11/2019	DGTI	Actualización CXF a versión 3.3.4
010	27/04/2020	DGTI	Eliminación referencias urls portales
011	23/08/2021	DGTI	Eliminación referencias a e-Sirca
012	15/03/2022	DGTI	Incorporación del punto Documentación a aportar

1.3 Estado del documento

Responsable aprobación	Fecha

2. Descripción general

2.1 Alcance

Este documento pretende ser una guía de usuario para la generación de servicios servidores mediante CXF y siguiendo el protocolo SCSP adoptado por la ***Plataforma Autonómica*** *de Interoperabilidad de la Comunitat Valenciana* (a partir de ahora PAI).

2.2 Objetivos

Los objetivos del presente documento son:

- Describir las particularidades y condicionantes de la generación de servicios servidores mediante CXF.

2.3 Audiencia

Nombre y Apellidos	Rol

Tabla 1: Audiencia

2.4 Glosario

Término	Definición
PAI	Plataforma Autonómica de Interoperabilidad de la Comunitat Valenciana

Tabla 2: Glosario

2.5 Referencias

Referencia	Título

Tabla 3: Referencias

3. Introducción

La Plataforma Autonómica de Interoperabilidad de la Comunitat Valenciana (PAI) pretende generar una guía de usuario para la generación de servicios web mediante la herramienta CXF .

Se ha de tener en cuenta que todo el proceso de instalación y configuración básica de los entornos no es objeto de este documento. Este documento solo relata los pasos que se han de realizar para el correcto funcionamiento de un servicio web creado con CXF cumpliendo las buenas prácticas.

4. Guía de para la generación de servicios web

4.1 Configuración y requisitos

El código fuente ejemplo que se comenta en esta sección se puede bajar desde el apartado "¿Cómo usar la plataforma?" del portal de la PAI, en el botón de "Ayuda al Desarrollador", el usuario y el password para acceder a dicho botón se debe solicitar al correo electrónico a esirca_ Interoperabilidad <esirca_interoperabilidad@gva.es>

Se trata de un proyecto ejemplo, que contiene los fuentes y un compilado preparado para ejecutar en Jboss como servidor de aplicaciones. Se han utilizado para realizarlo y testearlo, las siguientes tecnologías:

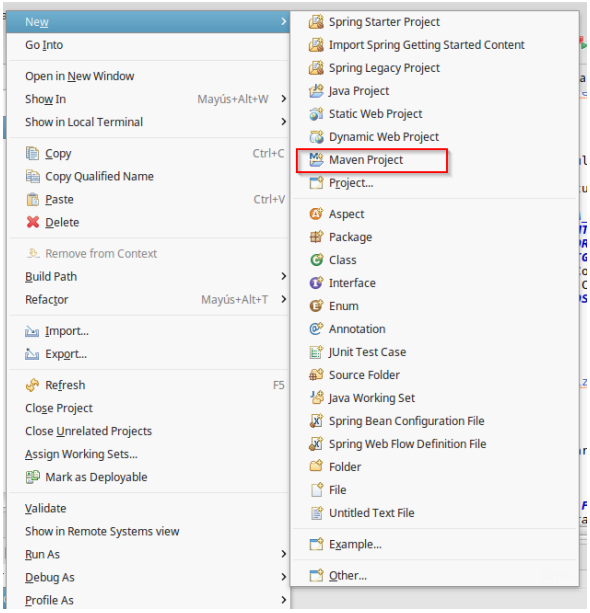
- String Tool Suite en su versión 3.8.4 (Spring 3.0.5)
  - http://spring.io/tools
- SoapUI : Herramienta para testeo de WebServices, usaremos la versión 5.3.0
  - http://www.soapui.org/
- CXF
  - http://cxf.apache.org/
- Maven 3.3.9

NOTA: Si la aplicación sigue el estándar de la oficina java, es de interés consultar la documentación expuesta en su espacio de confluence, en concreto la relativa a la capa de servicios (190-analisis-capa-servicios-soap).

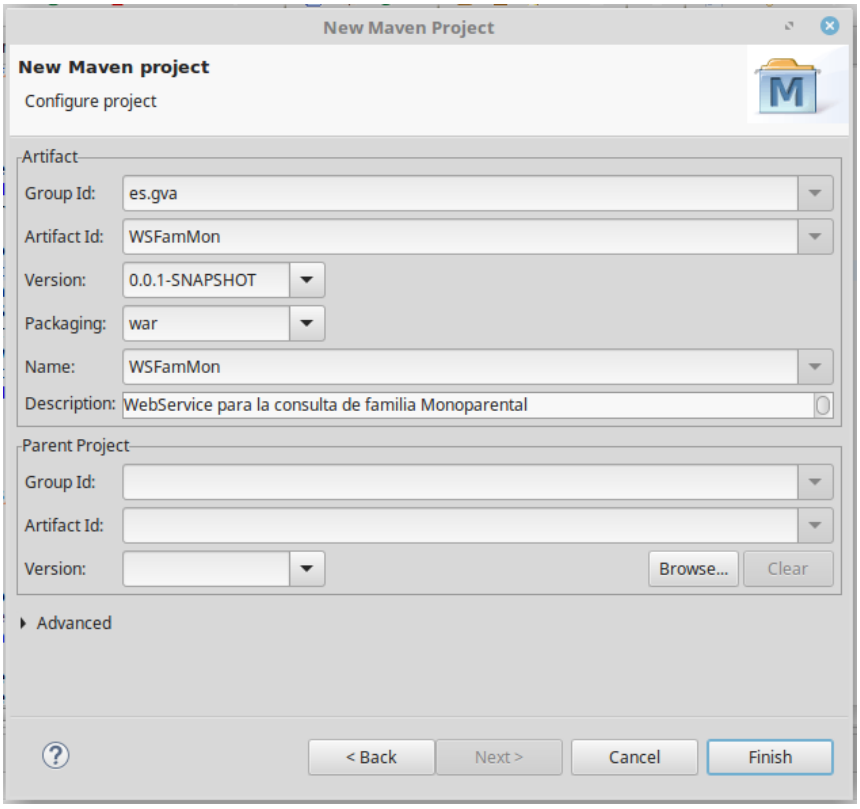
4.2 Generación de servicio web

4.2.1 Proyecto en Spring Tool Suite (STS)

Una vez descargado y ejecutado el STS, versión 3.8.4 desde la página http://spring.io/tools, nos dirigimos al menú y en el apartado nuevo seleccionamos proyecto tipo Maven.



Configuramos el proyecto con los datos necesarios.



4.2.2 Generación clases (Contract-First)

Es recomendable realizar la generación de clases del servicio a partir de un WSDL correctamente definido y que cumpla el documento de buenas prácticas difundido en el portal de la PAI, donde se

tratan puntos como la separación de esquemas del fichero de definición del servicios (WSDL) o la especificación de esquemas SCSP en caso de servicios de verificación, entre otros.

Por ello, una vez tengamos el WSDL ejecutaremos la instrucción de CXF que permite realizar la creación de las clases java. Existen diferentes formas de realizar esta creación:

- Plugin CXF de Eclipse: La versión del plugin, aunque funcional, no contempla la versión CXF 3
- Herramienta SoapUI: Es posible configurar CXF para la creación de los fuentes, aunque existen posibles problemas con los paquetes donde se generan estas clases y pueden generar conflictos.
- Comando wsdl2java: Ejecutando directamente el fuente tenemos total control sobre los parámetros.

Por ello, en este manual vamos a utilizar el comando wsdl2java que reside en el bin de CXF. Es interesante conocer los parámetros permitidos por el comando, que podemos consultar en http://cxf.apache.org/docs/wsdl-to-java.html.

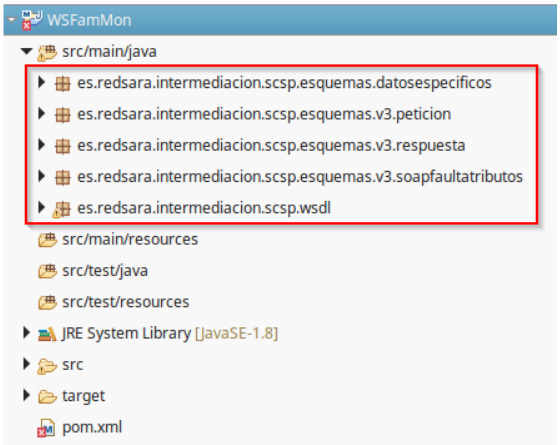
Nos dirigiremos pues al directorio bin de donde tengamos la instalación del CXF y ejecutaremos el siguiente comando:

```
wsdl2java -d [DIRECTORIO_DESTINO] -impl  [RUTA_WSDL]/[NOMBRE_WSDL].wsdl
```

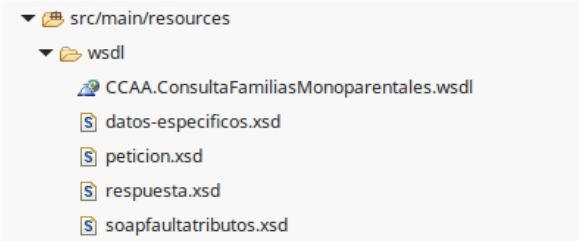
NOTA: Esta instrucción fallará en caso de que la definición del wsdl no sea correcta o no pueda encontrar los xsd asociados al mismo.

Tras la ejecución de esta instrucción y en el directorio destino, podemos encontrar la estructura de carpetas y las clases correspondientes creadas, junto a la implementación de las interfaces del servicio. Copiamos toda la estructura a nuestro proyecto, en el directorio *src/main/java*. Una vez realizada esta acción refrescamos el proyecto desde el STS.

Nota: el WSDL escogido como ejemplo es el de Familia Monoparental que cumple las especificaciones SCSP del MINHAP, definidos para los servicios de verificación de Datos en el ámbito de las Administraciones Públicas.



Seguidamente debemos incluir en la ruta *src/main/resources* el wsdl y los xsd.



Posteriormente, editar la implementación para indicar el wsdl de forma local consiguiendo de esta forma que se muestre correctamente. Para ello, nos vamos a *CCAAConsultaFamiliasMonoparentalesImpl.java* y modificamos el parámetro *wsdlLocation* de la anotación WebService indicando la ruta local "*/wsdl/CCAA.ConsultaFamiliasMonoparentales.wsdl"*

4.3 Configuración CXF y Spring

Tras realizar las acciones comentadas con anterioridad, procederemos a realizar la configuración deseada de CXF.

Primero añadiremos las dependencias deseadas al ***pom.xml***

```
<!-- CXF -->
<dependency>
	<groupId>org.apache.cxf</groupId>
	<artifactId>cxf-api</artifactId>
	<version>${cxf.version}</version>
</dependency>
<dependency>
	<groupId>org.apache.cxf</groupId>
	<artifactId>cxf-rt-ws-security</artifactId>
	<version>${cxf.version}</version>
</dependency>
<dependency>
	<groupId>org.apache.cxf</groupId>
	<artifactId>cxf-rt-frontend-jaxws</artifactId>
	<version>${cxf.version}</version>
</dependency>
<dependency>
	<groupId>org.apache.cxf</groupId>
	<artifactId>cxf-rt-transports-http</artifactId>
	<version>${cxf.version}</version>
</dependency>
<!-- Spring -->
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-web</artifactId>
	<version>${spring.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-context-support</artifactId>
	<version>${spring.version}</version>
```

```
            </dependency>
```

En el archivo ***web.xml*** situado en */src/main/webapp/WEB-INF* realizaremos la configuración del descriptor de despliegue.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
	<context-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>/WEB-INF/applicationContext.xml</param-value>
	</context-param>
	<listener>
		<listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
	</listener>
	<servlet>
		<servlet-name>CXFServlet</servlet-name>
		<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>CXFServlet</servlet-name>
		<url-pattern>/services/*</url-pattern>
	</servlet-mapping>
</web-app>
```

Como vemos creamos una referencia al fichero ***applicationContext.xml*** que contiene la configuración del contexto.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
	xmlns:jaxws="http://cxf.apache.org/jaxws"
xmlns:task="http://www.springframework.org/schema/task"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
	  http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
	  http://www.springframework.org/schema/context
	  http://www.springframework.org/schema/context/spring-context-3.2.xsd
	  http://www.springframework.org/schema/task
	  http://www.springframework.org/schema/task/spring-task-3.2.xsd
	  http://cxf.apache.org/jaxws
	  http://cxf.apache.org/schemas/jaxws.xsd">

	<jaxws:endpoint id="wsFamMon"

	implementor="es.redsara.intermediacion.scsp.wsdl.CCAAConsultaFamiliasMonoparentales
Impl"
		address="/CCAA.ConsultaFamiliasMonoparentales">
	</jaxws:endpoint>
</beans>
```

Donde podemos observar bajo la etiqueta *jaxws:endpoint* la configuración del endpoint donde encontramos la clase que lo implementa en el atributo *implementor* y la dirección de publicación bajo el atributo *address*. Como hijos de esta etiqueta incluiremos posteriormente los diferentes interceptores si se necesitasen.

Una vez realizada toda esta configuración, utilizaremos la Implementación (por ejemplo, *CCAAConsultaFamiliasMonoparentalesImpl.java*) para programar la respuesta dependiendo de la naturaleza del servicio web y parámetros de la petición.

4.4 Configuración JBOSS

En caso de realizar el despliegue sobre servidores Jboss, como es el caso, es recomendable sobreescribir el contexto por defecto, creando el fichero ***jboss-web.xml*** en *src/main/webapp/WEB-INF/*

```
<jboss-web>
	<context-root>/webservices/WSFamMon</context-root>
</jboss-web>
```

Tras realizar esta configuración, cuando iniciemos la aplicación el wsdl del servicio estará disponible en la dirección:

```
http://[Server][Port]/[Contexto]/services/[Nombre_Servicio]?wsdl
```

En este caso por ejemplo:

```
http://localhost:8081/webservices/WSFamMon/services/CCAA.ConsultaFamiliasMonoparentales?wsdl
```

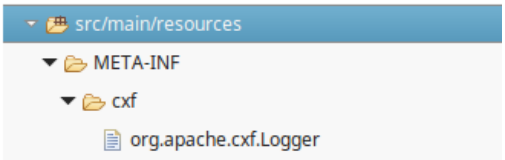
4.5 Configuración log4j

Para realizar la configuración e integración de la librería log4j en nuestro servicio web, es necesario realizar los siguientes pasos:

Creamos en ***jboss-deployment-structure.xml***

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
	<deployment>
		<exclusions>
			<module name="org.apache.log4j" />
		</exclusions>
	</deployment>
</jboss-deployment-structure>
```

Creamos el fichero ***org.apache.cxf.Logger*** en *src/main/resources/META-INF/cxf.*



Con el siguiente contenido:

```
org.apache.cxf.common.logging.Log4jLogger
```

Tras esta configuración podremos añadir los interceptores de log en applicationContext.xml de la siguiente forma:

```
	<bean id="loggingInInterceptor"
class="org.apache.cxf.interceptor.LoggingInInterceptor"
		parent="abstractLoggingInterceptor">
	</bean>
	<bean id="loggingOutInterceptor"
class="org.apache.cxf.interceptor.LoggingOutInterceptor"
		parent="abstractLoggingInterceptor">
	</bean>

	<bean id="abstractLoggingInterceptor" abstract="true">
		<property name="prettyLogging" value="true" />
	</bean>

	<jaxws:endpoint id="wsFamMon"

	implementor="es.redsara.intermediacion.scsp.wsdl.CCAAConsultaFamiliasMonoparentales
Impl"
		address="/CCAA.ConsultaFamiliasMonoparentales">
		<jaxws:inInterceptors>
			<ref bean="loggingInInterceptor" />
		</jaxws:inInterceptors>
		<jaxws:outInterceptors>
			<ref bean="loggingInInterceptor" />
		</jaxws:outInterceptors>
	</jaxws:endpoint>
```

Crear el fichero ***log4j.properties*** con el contenido de la configuración como por ejemplo:

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=[PATH_LOGS]/[APP_NAME].log

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p %c{1}:%L - %m%n
```

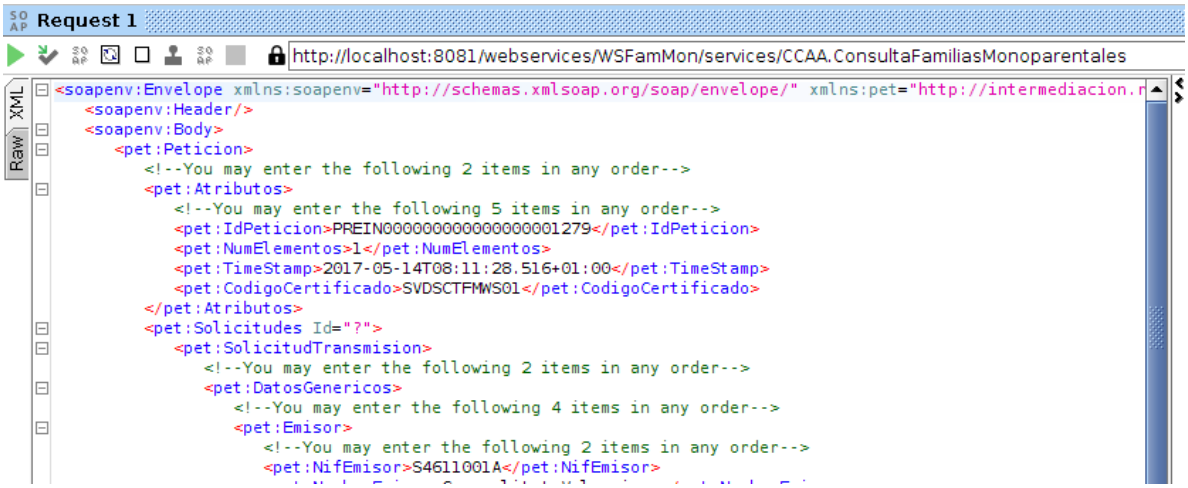
NOTA: En caso de necesitar cargar este fichero de forma externa para cumplir con la normativa de Sistemas GVA, es necesario incluir el siguiente extracto en el web.xml teniendo en cuenta que la variable de sistema ${asa.conf} existe en el servidor.

```
	<context-param>
		<param-name>log4jConfigLocation</param-name>
		<param-value>file:${asa.conf}/log4j.properties</param-value>
	</context-param>
	<listener>
		<listener-class>org.springframework.web.util.Log4jConfigListener</listener-
class>
	</listener>
```

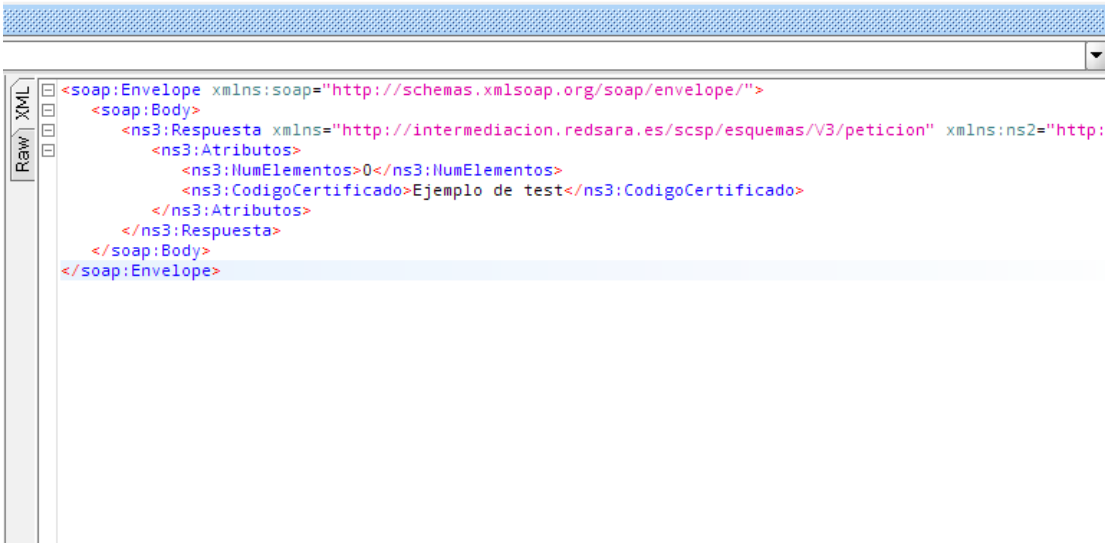
4.6 SOAPUI

Podemos utilizar SoapUI como herramienta para el testeo de nuestro servicio web, una vez desplegada la aplicación y accesible en el wsdl, podremos incluirlo en el SOAPUI para realizar peticiones de prueba y verificar la respuesta.

Petición:



Respuesta:



5. Documentación a aportar

El desarrollo de un servicio web debe ir acompañado de la generación de documentación, cuyo objetivo es facilitar la integración, tanto al cliente, como de cualquier Middleware como es la PAI.

En este escenario, se puede conseguir este objetivo con la entrega de dos documentos; Juego de prueba y contrato de integración.

5.1 Contrato Integración

Cuando nos referimos a contratos de integración en una arquitectura orientada a servicios, tendemos a pensar en las definiciones WSDL en el caso de los servicios SOAP. Este fichero extendido de XML, proporciona la información necesaria respecto a las operaciones, formato de mensajes y protocolos.

Pero, en la mayoría de las ocasiones, esta definición **no es suficiente,** ya que en las integraciones se necesita más información con respecto a la funcionalidad del servicio, modelo de comunicación y el detalle de los campos. Por ello, es necesario acompañar a las definiciones tipo WSDL de un documento de integración que permita llegar al detalle anteriormente comentado.

Estos documentos o contrato de integración contendrán al menos los siguientes puntos:

- Registro de versiones y cambios.
- Descripción del servicio
- Datos de acceso
- Definición de datos por operación
- Catálogo de errores posibles

Es posible contactar con el equipo de la PAI para obtener información sobre el formato del mismo.

5.2 Juegos de pruebas

La generación y entrega de un juego de pruebas del servicio a publicar, es una característica necesaria que permite la validación de la integración, tanto al futuro cliente, como a la propia PAI.

Generalmente estos documentos deben proporcionar información sobre los parámetros de entrada y respuestas esperadas en cada una de las operaciones.

Cuanto más completo sea un juego de pruebas, más ayudará en su publicación y su integración, pudiendo tratar flujos de invocación con errores esperados.

6. Anexo I. Añadir WS-Security Signature (Respuesta)

En este apartado vamos a tratar de explicar cómo añadir, en la respuesta del servicio, la cabecera de seguridad con ws-security mediante un certificado.

Para ello y tomando como base el proyecto desarrollado en los apartados anteriores, deberemos realizar varias acciones sobre el mismo.

En primer lugar, creamos el "***properties***" donde irán todas las propiedades del certificado que vayamos a utilizar en la firma del mensaje e incluir el mismo en una ruta accesible.

```
org.apache.ws.security.crypto.merlin.keystore.file=[PATH]/[CERT].JKS
org.apache.ws.security.crypto.merlin.keystore.password=[PASSWORD_JKS]
org.apache.ws.security.crypto.merlin.keystore.type=JKS
org.apache.ws.security.crypto.merlin.keystore.alias=[ALIAS_CERT]
org.apache.ws.security.crypto.merlin.keystore.alias.keypass=[PASSWORD_CERT]
```

Por otro lado, debemos incluir un interceptor en la salida del mensaje para que mediante una clase referenciada en el mismo cree la cabecera arreglo a los parámetros que le pasemos. Para ello nos dirigimos al fichero de propiedades de CXF y añadimos las siguientes propiedades.

```
	<jaxws:endpoint id="wsFamMon"

	implementor="es.redsara.intermediacion.scsp.wsdl.CCAAConsultaFamiliasMonoparentales
Impl"
		address="/CCAA.ConsultaFamiliasMonoparentales">
		<jaxws:inInterceptors>
			...
		</jaxws:inInterceptors>
		<jaxws:outInterceptors>
			...
			<ref bean="wss4jOutConfiguration" />
		</jaxws:outInterceptors>
	</jaxws:endpoint>

	<bean id="wss4jOutConfiguration"
class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
		<property name="properties">
			<map>
				<entry key="action" value="Signature" />
				<entry key="user"
value="${org.apache.ws.security.crypto.merlin.keystore.alias}" />
				<entry key="mustUnderstand" value="true" />
				<entry key="signaturePropFile"
value="serviceKeystore.properties" />
				<entry key="passwordCallbackClass"
					value="es.gva.interceptores.InterceptorFirma" />
				<entry key="signatureAlgorithm"
value="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
			</map>
		</property>
	</bean>
```

Como observamos, en la declaración del bean incluimos las propiedades de la firma de las cuales podemos destacar:

```
<entry key="action" value="Signature" />
```

Esta declaración informa de que acciones vamos a realizar en la firma. En este caso consideramos que añadir la firma.

```
<entry key="user" value="${org.apache.ws.security.crypto.merlin.keystore.alias}" />
```

Alias del keystore configurado que vamos a utilizar.

```
<entry key="mustUnderstand" value="true" />
```

Propiedad que informa de si incluiremos o no el parámetro "***mustUnderstand***" en la cabecera.

```
<entry key="signaturePropFile" value="serviceKeystore.properties" />
```

Enlazamos el ***properties*** donde va referenciado el jks en este caso.

```
<entry key="passwordCallbackClass"      value="es.gva.interceptores.InterceptorFirma" />
```

En este momento, indicamos la clase que va a manejar y realizar las acciones sobre el mensaje. Posteriormente describimos cual va a ser la implementación de la misma.

Debemos crear la clase manejadora "***Handler***" que actuará sobre el menaje, esta clase, extiende a '***CallbackHandler***' y en su constructor debemos hacer referencia al "***.properties***" creado anteriormente e indicarle los parámetros de alias y password con los cuales firmamos.

```
package es.gva.interceptores;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.ResourceBundle;

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

import org.apache.ws.security.WSPasswordCallback;

public class InterceptorFirma implements CallbackHandler {

	private Map<String, String> passwords =
			new HashMap<String, String>();

	public InterceptorFirma() {
		//Cargando *.properties del Keystore
		ResourceBundle rb = ResourceBundle.getBundle("serviceKeystore");
		//Accediendo al alias con el que se va a firmar

	passwords.put(rb.getString("org.apache.ws.security.crypto.merlin.keystore.a
lias"),rb.getString("org.apache.ws.security.crypto.merlin.keystore.alias.keypass"
));

	}
```

```
      public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
            for (int i = 0; i < callbacks.length; i++) {
                  WSPasswordCallback pc = (WSPasswordCallback)callbacks[i];

                  String pass = passwords.get(pc.getIdentifier());
                  if (pass != null) {
                        pc.setPassword(pass);
                        return;
                  }

            }
      }

}
```

Una vez hemos realizado estos pasos, podremos ejecutar el proyecto y comprobar mediante las herramientas de consumo que dispongamos, que el servicio creado está firmando las respuestas.

6.1 Encriptado de partes

Muchos de los servicios que se implementan realizan transferencias de datos sensibles, por ello, es interesante contemplar la posibilidad de encriptar parte de los mensajes, a cualquier nivel.

En este apartado vamos a tratar un ejemplo básico de como encriptar un campo del mensaje. Todas las modificaciones necesarias para ello se realizan en el fichero de configuración de cxf.

Tomamos la clase interceptor antes realizada y procedemos a ampliar las propiedades descritas dentro del bean.

```
<entry key="action" value="Timestamp Signature Encrypt"/>
```

Realizamos cambios en la action, añadiendo el valor "***Encrypt***"

```
<entry key="encryptionPropFile" value="serviceKeystore.properties"/>
<entry key="encryptionUser" value="[ALIAS_CERT]"/>
```

Declaramos el fichero y usuario con el que vamos a encriptar.

```
<entry key="encryptionParts"
value="{Content}{http://intermediacion.redsara.es/scsp/esquemas/V3/respuesta}CodigoCertificado"/>
<entry key="encryptionSymAlgorithm" value="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
<entry key="encryptionKeyTransportAlgorithm" value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
```

Y finalmente las partes que vamos a encriptar, teniendo en cuenta que se puede encriptar a nivel de contenido (Content) o elemento (Element), así como el algoritmo a utilizar. Recordamos que se encripta la primera entrada de la lista que se identifica con el campo y pertenece al namespace. Hay que tener en cuenta que la propiedad y el nombre son keysensitive

7. Anexo II. Añadir soapfaults tipo SCSP

En este anexo vamos a realizar las acciones correspondientes para añadir soapfaults al proyecto. Los pasos a seguir para resolver este caso son;

- Modificar el wsdl del servicio (entendiendo que disponemos del ***xsd*** que define los "***detail***" del **soapfault**)
- Regenerar las clases del servicio web, mediante el cambio en el ***pom*** y la instrucción antes utilizada.
- Programar la lógica en la implementación del servicio para lanzar el ***soapfault***.

Como primer paso, añadimos lo necesario al ***wsdl*** para que contemple los ***faults*** personalizados.

```
<wsdl:definitions
xmlns:ns3="http://intermediacion.redsara.es/scsp/esquemas/V3/soapfaultatributos">
```

Añadimos el namespace en la etiqueta "***definitions***", para ser utilizado posteriormente.

```
  <wsdl:types>
    <xsd:schema targetNamespace="http://intermediacion.redsara.es/xml-schemas">
            ...
            <xsd:import
namespace="http://intermediacion.redsara.es/scsp/esquemas/V3/soapfaultatributos"
schemaLocation="soapfaultatributos.xsd"/>
    </xsd:schema>
 </wsdl:types>
```

Dentro de los tipos importamos el ***xsd*** correspondiente al ***soapfault***, en este caso llamado soapfaultatributos.

```
<wsdl:message name="faultSCSPMessage">
    <wsdl:part element="ns3:Atributos" name="fault"/>
  </wsdl:message>
```

Añadimos un part con el elemento Atributos que figura en el ***xsd*** y lo nombramos como error.

```
 <wsdl:portType name="CCAAConsultaFamiliasMonoparentales">
    <wsdl:operation name="peticionSincrona">
      ...
        <wsdl:fault message="wsdl:faultSCSPMessage" name="FaultSCSP" />
    </wsdl:operation>
  </wsdl:portType>
```

En la interfaz portType añadimos una nuevo tipo para los ***fault***.

```
<wsdl:binding                    name="CCAAConsultaFamiliasMonoparentalesBinding"
type="wsdl:CCAAConsultaFamiliasMonoparentales">
```

```
    <soap:binding                                                    style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="peticionSincrona">
      <soap:operation soapAction="peticionSincrona" style="document"/>
      ...
        <wsdl:fault name="FaultSCSP">
          <soap:fault name="FaultSCSP" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
```

En la implementación de la operación añadimos el mismo tipo de recurso ***fault*** definido como "***literal***".

Con esto el ***wsdl*** está listo para hacer la operación ***wsdl2java*** de ***cxf*** que se realiza de la misma forma que hemos visto en punto 4.2.2.

La utilización de esta nueva funciona se realiza muy fácilmente con esta parte de código:

```
try {
    es.redsara.intermediacion.scsp.esquemas.v3.respuesta.Respuesta _return =
null;
    return _return;
} catch (java.lang.Exception ex) {
    es.redsara.intermediacion.scsp.esquemas.v3.soapfaultatributos.Atributos _atr
= new Atributos();
      _atr.setIdPeticion("50");
      throw new FaultSCSPMessage("Error",_atr);
}
```

Donde lanzamos la excepción del tipo generado por CXF a raíz del ***wsdl*** y completamos el esquema definido para pasárselo como parámetro.

NOTA: Encontraremos en el documento de "Desarrollo y consumo de servicios web. Buenas prácticas", en el portal de la PAI, los valores que se deben indicar en cada uno de los campos relativos al SoapFault y dependiendo de la naturaleza del servicio, sea verificación o instrumental.

8. Anexo III. SoapHeader mustunderstand

El atributo “mustUnderstand” se utiliza para indicar que el procesado del campo de la cabecera donde ha sido aplicado es obligatorio o no, en todos los destinos por los que pase y se procese el mensaje (Bus, Servicio Final,...).

Por ello, si se agrega mustUnderstand=”1” a un elemento de la cabecera como puede ser el nodo “Security”, se está indicando que el receptor de dicho mensaje debe reconocer ese elemento. Si no lo hace, se producirá un error en el procesamiento de la petición.

```
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" soap:mustUnderstand="1">
```

Ante un escenario donde el servicio reciba dicha cabecera, actúa de la siguiente forma; al recibir una petición, el módulo "SOAP binding" se encarga de recopilar la lista de las cabeceras que puede entender, llamando al método *“getUnderstoodHeaders()”* de todos los interceptores de entrada cargados en el flujo y cotejando las requeridas (mustUnderstand="1") con dicha lista.

```
{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd}Security
```

El escenario más habitual y cuya configuración por defecto en la creación de un cliente es que el atributo soap:mustUnderstand="1" se aplique a la cabecera “Security”. Por tanto se pueden aplicar 2 variantes para solucionar este escenario.

8.1 Creación Dummy incerceptor

Crear un interceptor personalizado, cuya función se reduzca a responder al "SOAP binding" que entiende las cabeceras WS-Security sin realizar ninguna actuación en su ejecución.

WSSPassThroughInterceptor.java

```
package es.gva.interceptores;

import java.util.HashSet;
import java.util.Set;

import javax.xml.namespace.QName;

import org.apache.cxf.binding.soap.SoapMessage;
import org.apache.cxf.binding.soap.interceptor.AbstractSoapInterceptor;
import org.apache.cxf.phase.Phase;
import org.apache.ws.security.WSConstants;

public class WSSPassThroughInterceptor extends AbstractSoapInterceptor {

  private static final Set<QName> HEADERS = new HashSet<QName>();
  static {
    HEADERS.add(new QName(WSConstants.WSSE_NS, WSConstants.WSSE_LN));
    HEADERS.add(new QName(WSConstants.WSSE11_NS, WSConstants.WSSE_LN));
    HEADERS.add(new QName(WSConstants.ENC_NS, WSConstants.ENC_DATA_LN));
  }

  public WSSPassThroughInterceptor() {
    super(Phase.PRE_PROTOCOL);
  }

  public WSSPassThroughInterceptor(String phase) {
```

```
    super(phase);
  }

  @Override
  public Set<QName> getUnderstoodHeaders() {
    return HEADERS;
  }

  public void handleMessage(SoapMessage soapMessage) {
  }
}
```

Como se puede observar en el código, se incluyen en la lista las cabeceras pertenecientes a WS-Security.

cxf-ScspWsGva.xml

```
<bean id="wssPassThroughInterceptor" class="es.gva.interceptores.WSSPassThroughInterceptor" />

        <jaxws:endpoint id="wsFamMon"
implementor="es.redsara.intermediacion.scsp.wsdl.CCAAConsultaFamiliasMonoparentalesImpl"
address="/CCAA.ConsultaFamiliasMonoparentales">
                <jaxws:inInterceptors>
                        ...
                        <ref bean="wssPassThroughInterceptor" />
                </jaxws:inInterceptors>
                ...
        </jaxws:endpoint>
```

Registramos en el fichero de configuración cxf el interceptor, creando el bean en relación al class. Posteriormente hay que incluir la referencia al bean en los interceptores cargados en la entrada, de esta forma se ejecutará ante una petición entrante.

Solo con estos cambios, las peticiones que contengan el atributo mustUnderstand van a ser procesadas correctamente por el servicio.

8.2 Configuración WSS4JInInterceptor

Utilizar el interceptor de entrada propio de WSS4j para poder procesar las peticiones con el atributo mustUnderstand requiere realizar alguna acción en dicho interceptor, por lo tanto, para escenarios con firmas SOAP WSS debemos indicar que se debe validar la firma de la petición y al igual que en escenario anterior incluirlo en el flujo de entrada.

```
        <jaxws:endpoint id="wsFamMon"
implementor="es.redsara.intermediacion.scsp.wsdl.CCAAConsultaFamiliasMonoparentalesImpl"
address="/CCAA.ConsultaFamiliasMonoparentales">
                <jaxws:inInterceptors>
                        ...
                        <ref bean="wss4jInConfiguration" />
                </jaxws:inInterceptors>
                ...
        </jaxws:endpoint>

        <bean id="wss4jInConfiguration" class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
                <property name="properties">
                        <map>
                                <entry key="action" value="Signature" />
                                <entry key="signaturePropFile" value="serviceKeystore.properties" />
                        </map>
```

```
            </property>
        </bean>
```

En este caso debemos tener en cuenta un factor importante para que las peticiones se procesen con éxito. Como vemos en el código, se debe configurar una entrada de configuración "*signaturePropFile*" que hace referencia al properties donde se configura el jks que se utilizará para verificar la cadena de certificación del certificado que firma la petición, así pues, en dicho jks se han de incluir las CA's que se quieren admitir.