Sistema Integral Multicanal de Atención al Ciudadano

e-SIRCA-Desarrollo y consumo de servicios web. Buenas prácticas

GENERALITAT VALENCIANA
CONSELLERIA DE HACIENDA Y ADMINISTRACIÓN PÚBLICA

Versión 012
Noviembre de 2014

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

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

Índice

Índice 2

1 Control del documento 4
- **1.1 Información general** 4
- **1.2 Histórico de revisiones** 4
- **1.3 Estado del documento** 4
- **1.4 Objetivos** 5
- **1.5 Audiencia** 5
- **1.6 Glosario** 5
- **1.7 Referencias** 5

2 ¿Por qué usar servicios web? 6
- **2.1 Servicios web en el ámbito de la Administración electrónica** 7

3 Criterios para crear un Servicios web 9
- **3.1 Granularidad de los servicios** 9
- **3.2 Definiendo el servicio (Contract-First)** 10
- **3.3 Norma Técnica** 10
  - 3.3.1 Estilo de construcción del servicio (binding style – operation use) 10
  - 3.3.2 Atributo de cabecera SOAP: MustUnderstand 11
  - 3.3.3 Definición de atributos y métodos legibles 12
  - 3.3.4 Recomendación respecto a definición de esquemas 14
    - 3.3.4.1 Patrón de diseño de XML Schema 14
    - 3.3.4.2 Representaciones del null en XML Schema 15
    - 3.3.4.3 Uso de los tags elementFormDefault y attributeFormDefault 15
    - 3.3.4.4 Espacio de nombres 17
- **3.4 Estrategias de versionado de servicios y XSD** 18
- **3.5 Definición de la Taxonomía de Servicio** 18
- **3.6 Servicios REST** 20

4 Entorno tecnológico de la PAI 21
- **4.1 Política de seguridad** 21
- **4.2 Proveedores de servicios** 21
- **4.3 Trazabilidad en el consumo de servicios de la PAI** 22
  - 4.3.1 Trazabilidad en los servicios instrumentales 22
  - 4.3.2 Trazabilidad en Servicios de Verificación 23
- **4.4 Definición de Errores de negocio y SoapFaults** 24
  - 4.4.1 Especificación Errores de negocio 24
  - 4.4.2 Especificación Soap Fault 25
    - 4.4.2.1 faultcode 25
    - 4.4.2.2 faultstring 26
    - 4.4.2.3 detail 26
    - 4.4.2.4 Ejemplo de SoapFault 28

5 Apoyo a los desarrolladores..... 30

5.1 Esqueleto de servicio web..... 30

5.1.1 Como utilizar gvNIX en el ámbito de la PAI..... 30

5.1.2 Bloques que componen el esqueleto..... 30

5.2 Clientes de servicios..... 31

1 Control del documento

1.1 Información general

Título	Desarrollo y consumo de servicios web. Buenas prácticas
Creado por	DGTI
Revisado por	
Lista de distribución	
Nombre del fichero	Desarrollo_y_consumo_de_servicios_web_Buenas_prácticas_v12.odt

1.2 Histórico de revisiones

Versión	Fecha	Autor	Observaciones
001	13/02/2012	DGM	Versión Inicial
002	08/04/2014	DGTI	Inclusión particularidades PAI
003	24/04/2014	DGTI	SoapFaults y gestión de errores
004	1/05/2014	DGTI	Eliminar texto/puntos generales y ambiguos. Añadir criterios técnicos que deben implementar los desarrolladores
005	16/05/2014	DGTI	Modificación e inclusión de nuevos tags y revisión general del documento.
006	10/09/2014	DGTI	Revisión del documento. Inclusión faultcode.
007	23/09/2014	DGTI	Reorganización del contenido. Inclusión de punto para proveedores de servicios y espacios de nombres.
008	26/09/2014	DGTI	Revisión del contenido. Reorganización de los puntos. Inclusión de Ayuda a los desarrolladores, patrones XML Schema, ¿Por qué usar servicios web?
009	31/10/2014	DGTI	Revisión de contenido. Sustituidas referencias a PIE (Plataforma de Intermediación del Estado) por PID (Plataforma de Intermediación de Datos). Referencia al documento de generación de servicios web.
010	21/11/2014	DGTI	Revisión del contenido. Añadido MustUnderstand. Cambio en Granularidad.
011	28/11/2014	DGTI	Modificaciones en faultcode, soapfault, definición de atributos y métodos, granularidad de servicios, parámetros de los xsd.
012	01/12/2014	DGTI	Revisión del contenido. Cambio en ejemplo soapFault, reasignación de puntos para la norma técnica.

1.3 Estado del documento

Responsable aprobación	Fecha

1.4 Objetivos

El presente documento pretende guiar en las buenas prácticas en el desarrollo de servicios web para permitir orientar la plataforma de servicios SOA de la GVA hacia un entorno de interoperabilidad real y eficiente.

1.5 Audiencia

Nombre y Apellidos	Rol
	CONSUMIDORES Y DESARROLADORES DE SERVICIOS

Tabla 1: Audiencia

1.6 Glosario

Término	Definición

Tabla 2: Glosario

1.7 Referencias

Referencia	Título

Tabla 3: Referencias

2 ¿Por qué usar servicios web?

SOA (Service Oriented Architecture) es una arquitectura de Software orientada a servicios. Para implementar con éxito una SOA se consideran unos principios importantes sobre los que debemos trabajar. Sobre ellos hemos destacado los que nos parecen más importantes y por tanto el "core" SOA:

- **Bajo acoplamiento**: Es uno de los factores críticos para el éxito de la estrategia SOA. El objetivo del bajo acoplamiento es reducir las dependencias entre los sistemas implicados. En el caso de los Web services la comunicación entre consumidores y servicios se realizará mediante la interfaz del servicio (WSDL), logrando así la independencia entre el servicio a ejecutar y el consumidor.
- **Reusabilidad**: Un servicio debe ser diseñado y construido pensando en su reutilización dentro de la misma aplicación, empresa o dominio público para su uso masivo. En el caso de los Servicios Web, tienen que ver en este apartado la relación entre WSDL y XML Schemas, como veremos más adelante el hecho de separar los XSD del WSDL es clave para su reutilización.
- **Descubrimiento**: Todo servicio debe poder ser descubierto de alguna forma para que pueda ser utilizado, consiguiendo así evitar la creación accidental de servicios que proporcionen las mismas funcionalidades.
- **Interoperabilidad**: Permite que los consumidores y los servicios desarrollados en tecnologías y plataformas diferentes puedan información y colaborar.
- **Gobernabilidad**: En un entorno SOA, la gobernanza guía el desarrollo de servicios reutilizables, lo que establece cómo se diseñarán y se desarrollarán los servicios y cómo cambiarán con el tiempo. Establece acuerdos entre los proveedores de servicios y los consumidores de esos servicios y indicando a los consumidores lo que deben esperan y lo que los proveedores están obligados a proporcionar.

Si bien SOA no está estrictamente vinculado necesariamente con Servicios Web, estos suman cada día más importancia ya que las características que poseen los convierten en el mecanismo ideal para cubrir los principios de la orientación a servicios (SOA).

Básicamente los Servicios Web son mecanismos de comunicación que permiten interoperabilidad entre aplicaciones a través del uso de estándares abiertos. De esta manera aplicaciones desarrolladas en diferentes lenguajes de programación y ejecutadas en

diferentes plataformas pueden intercambiar datos y presentar información dinámica al usuario de forma homogénea y coherente.

2.1 Servicios web en el ámbito de la Administración electrónica

En el entorno del desarrollo de servicios web en el ámbito de Administración electrónica, hay que tener muy en cuenta la Resolución de 28 de Junio de 2012, de la Secretaría de Estado de Administraciones Públicas, por la que se aprueba la Norma Técnica de Interoperabilidad de Protocolos de intermediación de datos. En dicha resolución se especifica que de forma general, en servicios de intercambio e intermediación, se debe seguir la estructura del protocolo SCSP (Sustitución de Certificados en Soporte Papel) cuya especificación está disponible en el Portal de Administración electrónica PAE/CTT en la dirección http://administracionelectronica.gob.es/es/ctt/scsp.

La especificación SCSP define un conjunto de mensajes, basados en XML, definidos por sus correspondientes esquemas XSD.

El sistema permite que la petición-respuesta sea síncrona o asíncrona y los mensajes van firmados mediante WS-Security.

- Mensajes generales: Son comunes a todos los servicios y están compuestos por los siguientes mensajes.
  - Petición (peticion.xsd) con namespace
  - Confirmación (confirmacionPeticion.xsd) con namespace
  - Solicitud de Respuesta (solicitudRespuesta.xsd) con namespace
  - Respuesta (respuesta.xsd) con namespace
  - Soap Fault Atributos (soapfaultatributos.xsd) con namespace
- Mensajes Específicos: Depende de cada certificado administrativo o transmisión de datos. Lo define el organismo emisor.
  - Datos Específicos con namespace

La PAI dispone de todos los esquemas definidos en el ENI.

La recomendación desde la PAI es seguir estos esquemas en lo que a servicios de verificación e intermediación compete, para facilitar la homogeneidad en el consumo de servicios ya sea como cedentes o requirentes de información en el ámbito de la norma técnica de interoperabilidad. En el caso de servicios denominados como instrumentales se han de seguir en la medida de lo posible las especificaciones y normas expuestas en este documento, pudiendo en caso de duda consultar a los responsables de la PAI en el uso de los estándares en base a los servicios a publicar.

Al tratar en la mayoría de ocasiones, de información sensible a LOPD, se deberán tener muy presentes las medidas en cuanto la confidencialidad e integridad de la información

intercambiada, que será protegida conforme al grupo de «medidas de protección», capítulos «Protección de las comunicaciones» (mp.com) y «Protección de la información» (mp.info) definidas en el Real Decreto 3/2010, de 8 de enero, y con las medidas de seguridad dispuestas en la Ley Orgánica 15/1999, de 13 de diciembre, y normativa de desarrollo, asegurando que no se almacena información personal de ningún ciudadano.

En el apartado III.6 de la Resolución de 28 de Junio, por el que se aprueba la Norma Técnica de Interoperabilidad de Protocolos de intermediación de datos, se hace referencia a los requisitos de trazabilidad y auditoría de los intercambios.

3 Criterios para crear un Servicios web.

Un Web Service es una parte del software que puede comunicarse con otra aplicación a través de una red usando un juego específico de protocolos estandarizados- SOAP, UDDI, WSDL.

Entre los principales beneficios que se exponen al hablar de los servicios Web, generalmente se encuentran aquellos que tienen que ver con granularidad e interoperabilidad, es decir, con la posibilidad de desarrollar componentes de software totalmente independientes que tienen funcionalidad propia, pero que son capaces de exponer tal funcionalidad y compartirla con otros servicios y aplicaciones para lograr crear sistemas más complejos.

Por otro lado, la visión planteada por este paradigma computacional, donde “todo es un servicio”, permite manejar un esquema de integración universal en el cual se pueden aprovechar todos los beneficios de cada componente con un nuevo nivel de complejidad y dinamismo.

Los Servicios Web permiten que las organizaciones integren sus diferentes aplicaciones de una manera eficiente, sin preocuparse por cómo fueron construidas, donde residen, sobre qué sistema operativo se ejecutan o cómo acceder a ellas.

3.1 Granularidad de los servicios

La granularidad de un servicio en un contexto SOA viene determinada por la cantidad global de funcionalidad encapsulada en dicho servicio. La granularidad de los servicios es fundamental para la reusabilidad y versatilidad de los servicios que se diseñen.

Los servicios web son definidos en WSDL (WebService Description Language), que de forma resumida, aglutinan en distintos métodos u operaciones, la funcionalidad implementada en los mismos. La recomendación desde la PAI es el desarrollo de un mayor número de servicios con menor número de métodos (servicios más simples), más que la encapsulación en un solo servicio de toda la funcionalidad aplicable a un determinado modelo de negocio complejo. Esto permite la clara identificación de los servicios de negocio y la simplificación en el consumo de los mismos debido a la minimización en las interacciones del método de consumo. Además permiten un mayor detalle en el registro de estadísticas de consumo de servicios.

Para entender mejor este concepto de granularidad, imaginemos un determinado organismo que por ejemplo, desarrolla un servicio que permite verificar los datos catastrales, pero al que

se le puede preguntar bien por bienes inmuebles asociados a un titular, bien por todos los datos catastrales de los inmuebles, que figuran en la base de datos del Catastro, asociados a un titular . La recomendación sería exponer ambas funcionalidades como servicios distintos.

Como resumen de este punto, se deben diseñar servicios web con las responsabilidades bien repartidas, que sean cohesivos, extensibles, escalables y reutilizables.

3.2 Definiendo el servicio (*Contract-First*)

Existen dos formas tradicionales de acometer el desarrollo de un servicio web. En la jerga tradicionalmente se les conoce como *ContractFirst* (contrato al principio) y *ContractLast* (contrato al final). La recomendación en este caso es claramente utilizar *ContractFirst,* es decir, definir las operaciones, métodos y datos del negocio del servicio como fase inicial del análisis, para implementar posteriormente el código. Los diseños Contract-First permiten dar mayor robustez al servicio frente a variaciones. También mejora los aspectos de reusabilidad, rendimiento y versionado, haciendo que sea una de las características más habituales en el diseño de webservices.

Se debe tener en cuenta que el desarrollo de servicios con esta filosofía requiere un planteamiento y definición iniciales, tanto de los servicios, como de los datos que van a formar parte del mismo. Este esfuerzo en la definición redundará como hemos dicho en la mejora del servicio a proporcionar, una mayor claridad en el planteamiento y una independencia en cuanto a la lógica de negocio interna de la aplicación.

3.3 Norma Técnica

En este capítulo se tratan puntos relativos a la aplicación de buenas prácticas en la definición técnica de los servicios web.

La definición de los mismo se basa en un conjunto de especificaciones y buenas prácticas definidas por la industria conocido como WS-I Basic Profile, para la obtención de servicio web interoperables.

3.3.1 Estilo de construcción del servicio (*binding style – operation use*)

En el capítulo de definición de los contratos de integración de los servicios, el documento en el que se especifican las operaciones y métodos está basado en el conocido WSDL. Respecto al WSDL, la particularidad más importante a tener en cuenta hace referencia a dos parámetros que enlazan cómo un determinado protocolo de mensajería es asociado a las operaciones y métodos del servicio (*binding style*) así como se conforman formalmente las peticiones (*operation use*). Las opciones posibles para el parámetro '*binding'* son 'rpc/document' mientras que para el parámetro '*use'* son: 'encoded/literal'. De las cuatro combinaciones posibles de

estos parámetros, la opción adecuada para publicar un servicio en el bus de la PAI exige que esta definición sea:

- binding=***document*** use=***literal***

La razón fundamental para el uso de esta opción es la ventaja que supone para los desarrolladores frente a cambios del interfaz del servicio. La simple adición de un parámetro a la operación de un servicio suele redundar en cambios importantes en el código desarrollado en base a un interfaz *rpc/encoded*, mientras que dichos cambios pueden ser menores en la evolución de un servicio codificado en base a un interfaz *document/literal*. Esta razón, entre otras, ha concluido finalmente que el style/operation *document/literal* se haya convertido en el enfoque de diseño recomendado en el diseño de webservices.

Mostramos a continuación un detalle que muestra sobre un determinado wsdl los parámetros sobre los que hacemos referencia anteriormente.

```
<wsdl:binding name="AutenticacionArangiPortTypeSoap11" type="tns:AutenticacionArangiPortTy
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" xmlns:sc
  <wsdl:operation name="autenticaUsuarioWS" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <soap:operation soapAction="" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"/>
    <wsdl:input name="autenticaUsuarioWSRequest" xmlns:wsdl="http://schemas.xmlsoap.org/ws
      <soap:body use="literal" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"/>
    </wsdl:input>
```

Finalmente, el WSDL debe ser compatible con el WS-I Basic Profile.

3.3.2 Atributo de cabecera SOAP: MustUnderstand

Las cabeceras en el ámbito SOAP son utilizadas para extender capacidades como puedan ser seguridad, enrutamiento, autenticación, transaccionabilidad, etc.

Los contratos de integración en el mundo SOAP (los wsdl) nos dicen claramente cómo debe ser un el cuerpo (*body*), qué operaciones debe tener un servicio, como se deben conformar las peticiones, etc. Pero nunca dicen nada acerca de cómo deben ser los headers, la propia especificación SOAP tampoco lo define, sino que las deja a expensas de las definiciones del usuario.

Cuando alguien manda una petición con el parámetro de cabecera *'mustUnderstand=1'* lo que está queriendo decir es que el nodo SOAP al que está mandando la petición debe obligatoriamente entender todos y cada uno de los parámetros de la cabecera. Si alguno no lo entiende, no puede ignorarlo, y el servicio web debe devolver un SoapFault informando de ello.

Pongamos un ejemplo para entender su utilidad; Imaginemos que una determinada petición exigiera que fuera transaccional, y que sólo se hiciera el commit si todas y cada una de las 'solicitudes' que viajan en la petición fueran exitosas. Si un servidor omitiera (no hiciera caso) el hecho de que el cliente le está informando que debe entender que esa petición exige transaccionalidad, podría darse el caso que la aplicación se quedara en un estado inconsistente. Por eso en la gran mayoría de las ocasiones se omite informar este parámetro, ya que exige que el nodo al que le están mandando la petición entienda y procese toda la cabecera adecuadamente.

En el escenario de la PAI, el *mustUnderstand* no tiene sentido ya que los consumidores de los servicios web no deben exigir el procesado de las cabeceras en los distintos nodos SOAP.

3.3.3 Definición de atributos y métodos legibles

No se permiten servicios que expongan métodos en los que un archivo XML lleve la lógica asociada como un solo parámetro de llamada al método, ya que en esta práctica encontramos ciertas desventajas.

Petición:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:es:gva:comunicaciones:ComunicacionesWS">
     <soapenv:Header/>
     <soapenv:Body>
          <urn:getComunicaciones>
               <!--Optional:-->
               <data><![CDATA[<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
                              <REQUEST_CONSULTA_COMUNICACIONES
xmlns="urn:gva:es:comunicaciones:requestConsultaComunicaciones">
<CODIGO_ENTIDAD>L046111-00000000</CODIGO_ENTIDAD>
<NUMERO_IDENTIFICACION>00265889D</NUMERO_IDENTIFICACION>
<ACUSE>true</ACUSE>
                              <PAGINACION>
<TAMANYO_PAGINA>1000</TAMANYO_PAGINA>
<NUMERO_PAGINA>0</NUMERO_PAGINA>
                              </PAGINACION>
                              </REQUEST_CONSULTA_COMUNICACIONES> ]]>
               </data>
          </urn:getComunicaciones>
     </soapenv:Body>
</soapenv:Envelope>
```

Esquemas:

```
<xsd:element name="getComunicaciones" type="getComunicaciones"/>
<xsd:complexType name="getComunicaciones">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="data" type="xs:string"/>
                </xsd:sequence>
</xsd:complexType>
```

Como vemos en el ejemplo anterior, la inclusión de un XML sin estandarizar en un parámetro definido en los esquemas como "*string*" en la petición tiene a aumenta la complejidad de la capa de negocio al tener que interpretar dicho XML incrustado.

En cuanto a la parte del consumidor, dificulta la comprensión del servicio al no estar basado en un estándar público, siendo más complicado el testeo y posterior tratamiento del mensaje en la capa de negocio de la aplicación.

Además a nivel de orquestación en la PAI, este tipo de prácticas hace imposible el tratamiento del mensaje transportado, así como la validación del mismo contra esquemas.

Para evitar esta problemática se debe eludir este tipo de prácticas en los servicios web y realizar la definición de esquemas precisos para cada uno de los elementos que van a viajar en la petición, a continuación vemos una aproximación de cómo deberían ser las peticiones y los esquemas bien estructurados.

Petición:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:es:gva:comunicaciones:v3:ComunicacionesWS">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:getComunicaciones>
         <urn:peticion>
            <urn:codigoEntidad>GV</urn:codigoEntidad>
            <urn:fechaFin>2016-02-16T17:30:04.347+01:00</urn:fechaFin>
            <urn:paginacion>
               <urn:tamanyoPagina>2</urn:tamanyoPagina>
               <urn:numeroPagina>1</urn:numeroPagina>
            </urn:paginacion>
         </urn:peticion>
      </urn:getComunicaciones>
   </soapenv:Body>
</soapenv:Envelope>
```

Extracto de esquema:

```
<xs:complexType name="consultaComunicacionesRequest">
            <xs:sequence>
                  <xs:element name="codigoEntidad" type="types:typeCodigoEntidad" />
                  <xs:element name="numeroComunicacion" type="types:typeNumeroComunicacion"
                        minOccurs="0" />
                  <xs:element name="numeroIdentificacion" type="types:typeNumeroIdentificacion"
                        minOccurs="0" />
                  <xs:element name="fechaInicio" type="xs:dateTime"
                        minOccurs="0" nillable="true" />
                  <xs:element name="fechaFin" type="xs:dateTime" minOccurs="0"
                        nillable="true" />
```

```
                <xs:element name="acuse" type="xs:boolean" minOccurs="0"
                      nillable="true" />
                <xs:element name="estados" minOccurs="0" nillable="true">
                      <xs:complexType>
                            <xs:sequence>
                                  <xs:element name="estado"
type="types:typeEstadoComunicacion"
                                        minOccurs="0" nillable="true" maxOccurs="unbounded" />
                            </xs:sequence>
                      </xs:complexType>
                </xs:element>
                <xs:element name="paginacion" type="tns:paginacion"
                      minOccurs="0" nillable="true" />
          </xs:sequence>
     </xs:complexType>
```

3.3.4 Recomendación respecto a definición de esquemas

Una recomendación habitual en la definición de servicios web es la de separar el modelo de datos de las distintas peticiones (*XSD's*) de las correspondientes definiciones asociadas a los métodos y operaciones del servicio (WSDL). Aunque es muy habitual el uso autocontenido en los wsdl de los modelos de datos, la separación de ambos ayuda en la organización y legibilidad del servicio, reduce el tamaño/complejidad del WSDL, permite utilizar editores especializados para el diseño del schema XSD y permite reutilizar schemas y namespaces.

A la hora de diseñar el schema XSD, conviene crear tipos y elementos globales (a nivel raíz) para poder reutilizarlos, tanto a nivel de elementos XML como clases del lenguaje de implementación del servicio web.

3.3.4.1 Patrón de diseño de XML Schema

Existen muchas maneras de diseñar XML Schemas válidos, pero si los queremos utilizar junto a los descriptores de contratos de servicios web (WSDL) , tendrán que cumplir además con las especificaciones WS-I Basic Profile con el propósito de cumplir unos requisitos de interoperabilidad y asegurar la compatibilidad en las invocaciones entre los mismos.

Las buenas prácticas en el diseño de XML Schema, giran en gran parte, en torno a la existencia de:

- Declaración de elementos (*element*) globales (elemento global es cualquier hijo de un nodo *<schema>* o una referencia directa a uno de ellos) o locales (Se anidan dentro de la estructura del *<schema>* y no son hijos directos de la raiz).
- Ídem para con la declaración de tipos definidos (*complexType* ó *simpleType*)

Existen varios patrones de diseño aceptados, entre los cuales se recomienda el uso del conocido como "*Salami Slice*" que consiste declarar los elementos a nivel global del documento y

utilizar las referencias a los mismos en los tipos complejos, facilitando la reutilización de componentes.

3.3.4.2 Representaciones del null en XML Schema

Hay dos maneras cuya utilización recomienda el WS-I Basic Profile para representar un valor nulo para elementos: ya sea con el atributo *nillable=true*, o con *minOccurs=0*.

Para matizar sobre su uso, se debe utilizar *minOccurs=0* para definir un elemento como opcional y *nillable=true* para indicar que un elemento puede ser vacío pero siempre estará presente en el mensaje XML.

3.3.4.3 Uso de los tags elementFormDefault y attributeFormDefault

En ocasiones cuando procesamos mensajes en nuestras aplicaciones de integración encontramos que algunas acciones especificadas como mapas de transformación o asignaciones *xpath()* no son ejecutadas como se esperan, después de revisar los esquemas de definición y los mensajes definidos no encontramos aparente lógica en la causa del error ya que los nombres de nodos (*xs:element*), atributos (*xs:attribute*) y nombres de espacios (xs:xmlns) utilizados están correctos.

Los parámetros *elementFormDefault* y *attributeFormDefault* especifican cómo deben ser formados los correspondientes XML, a nivel de elementos/nodos y a nivel de atributos de los elementos/nodos. Estas definiciones están muy relacionadas con los conceptos de elementos *globales* y *locales* en el XSD. En resumen un elemento global es cualquier hijo de un nodo nodo *schema* o una referencia directa a uno de ellos. Un elemento local es cualquiera que no sea global.

En el siguiente ejemplo mostramos detalle de ambos:

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <schema xmlns="http://www.w3.org/2001/XMLSchema"
 3:         targetNamespace="http://www.intertech.com/band"
 4:         xmlns:tns="http://www.intertech.com/band"
 5:         elementFormDefault="qualified"
 6:         attributeFormDefault="unqualified">
 7:
 8:     <complexType name="Band_Member_Type">
 9:         <sequence>
10:             <!-- LOCAL: not an immediate child of "schema" -->
11:             <element name="fullName" type="string" />
12:
13:             <!-- GLOBAL: a ref to an immediate child of "schema" -->
14:             <element ref="tns:instrument" maxOccurs="unbounded" />
15:         </sequence>
16:     </complexType>
17:
18:     <!-- GLOBAL: an immediate child of "schema" -->
```

```
19:     <element name="instrument" type="string" />
20:
21:     <!-- GLOBAL: an immediate child of "schema" -->
22:     <element name="bandName" type="string" />
23:
24:     <!-- GLOBAL: an immediate child of "schema" -->
25:     <element name="band">
26:         <complexType>
27:             <sequence>
28:                 <!-- GLOBAL: a ref to an immediate child of "schema" -->
29:                 <element ref="tns:bandName" />
30:
31:                 <!-- LOCAL: not an immediate child of "schema" -->
32:                 <element name="member" type="tns:Band_Member_Type" minOccurs="1"
33:                         maxOccurs="unbounded" />
34:                 </sequence>
35:             <attribute name="id" type="string" />
36:         </complexType>
37:     </element>
38: </schema>
```

Así pues, *elementFormDefault=qualified,* significa que todos los elementos (globales y locales) deben ser 'qualified', esto es, deben ser asociados a un determinado namespace. Por otra parte, *elementFormDefault=unqualified* significa que sólo los elementos globales deben ser asociados con namespaces, no así los elementos llamados 'locales'. Esto mismo se puede aplicar al parámetro *attributeFormDefault* el cual aplicará la norma a los nodos *<xs:attribute>.*

La recomendación es emplear *elementFormDefault* '***qualified' y*** *attributeFormDefault* "***unqualified***", para evitar colisiones y conflictos en caso de indefiniciones adecuadas de *namespaces*.

Un ejemplo de esta definición lo mostramos seguidamente:

```
<xs:schema targetNamespace="http://intermediacion.redsara.es/scsp/esquemas/V3/peticion" element-
FormDefault="qualified" attributeFormDefault="unqualified" xmlns="http://intermediacion.red-
sara.es/scsp/esquemas/V3/peticion" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://intermediacion.redsara.es/scsp/esquemas/datosespecificos">
      <xs:element name="Apellido1">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:maxLength value="40"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>
      <xs:element name="Apellido2">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:maxLength value="40"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>

.....

</xs:schema>
```

Por tanto la petición a este servicio seria correcta:

```
<soapenv:Body>
      <pet:Peticion>
            <pet:Atributos>
            ...
            </pet:Atributos>
            <pet:Solicitudes Id="">
                  <pet:SolicitudTransmision>
                        <pet:DatosGenericos>
                              <pet:Emisor>
                        ...
                              </pet:Emisor>
                              <pet:Solicitante>
                                    ...
                              </pet:Solicitante>
                              <pet:Titular>
                                    <pet:TipoDocumentacion>NIF</pet:TipoDocumentacion>
                                    <pet:Documentacion>99999999R</pet:Documentacion>
                                    <pet:Nombre>JUAN</pet:Nombre>
                                    <pet:Apellido1>ESPAÑOL</pet:Apellido1>
                                    <pet:Apellido2>ESPAÑOL</pet:Apellido2>

                              </pet:Titular>
                              <pet:Transmision>
                                    ...
                              </pet:Transmision>
                        </pet:DatosGenericos>
                  </pet:SolicitudTransmision>
            </pet:Solicitudes>
      </pet:Peticion>
</soapenv:Body>
```

Como podemos ver en la petición, con el *elementFormDefault=qualified* todos los elementos globales deben estar asociados a un determinado namespace. Seria además posible obviar las referencias en este caso a **pet** simplemente utilizando lo siguiente en el nodo Petición:

```
<Peticion xmlns="http://intermediacion.redsara.es/scsp/esquemas/V3/peticion">
```

3.3.4.4 Espacio de nombres

En XML, como en muchos lenguajes de programación, frecuentemente se debe especificar un "espacio de nombres" para varios elementos y atributos. Esto permite distinguir entre elementos que tienen el mismo nombre, pero diferentes propósitos y tal vez diferentes orígenes. XML hace referencia a un espacio de nombres mediante un identificador URI que no tiene por qué ser accesible. Por ejemplo, el espacio de nombres del esquema XML es http://www.w3.org/2001/XMLSchema. Para facilitar la tarea, sin embargo, también se asigna un alias o un prefijo. Por ejemplo, justo en el ejemplo anterior el espacio de nombres de esquema tiene un prefijo “xs:“. Recuerde que el alias es tan solo eso: un alias. El identificador URI es que realmente importa.

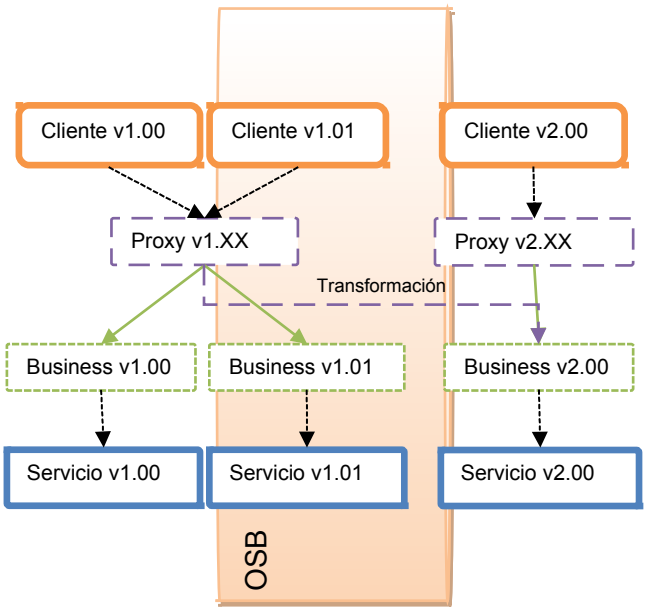
En el caso de los servicios de verificación se debe seguir la definición de espacio de nombres del protocolo SCSP (Sustitución de Certificados en Soporte Papel) cuya especificación está disponible en el Portal de Administración electrónica PAE/CTT en la dirección http://administracionelectronica.gob.es/es/ctt/scsp.

3.4 Estrategias de versionado de servicios y XSD

Por necesidades de los proyectos, los servicios web se encuentran en constante cambio y evolución. Las bases y principios de una arquitectura orientada a servicios (SOA) deben asegurar que los proveedores de servicios puedan:

- Liberar una nueva versión de un servicio, después de que los consumidores del servicio hayan probado y certificado la nueva versión del servicio.
- Liberar una nueva versión de un servicio, sin esperar a que los consumidores adapten la invocación al nuevo servicio.

En la PAI no se aceptarán escenarios de *transformación* como el que se muestra en la imagen, puesto que implica tener un conocimiento del negocio final, que en la mayoría de los casos se salen de las funciones de intermediación que se atribuye al bus de la PAI.



La recomendación, es que las distintas revisiones de los servicios guarden compatibilidad con versiones anteriores, para producir el menor número de incidencias en el consumo de las nuevas versiones.

3.5 Definición de la Taxonomía de Servicio

Con el fin de tener una correcta estructuración de los servicios incorporados a la infraestructura SOA de la PAI es de vital importancia mantener una correcta taxonomía de servicios, que permita categorizar dichos servicios a publicar conforme a criterios diferentes

que sirvan como base en la definición de alertas, monitorización, gestión de recursos asociados, etc.

La información que normalmente se requiere para categorizar estos servicios hace referencia a características relativas:

- Semántica e información del servicio. En la mayoría de los casos esta información viene auto contenida en los *wsdl* y *xsd* del servicio, pero en ocasiones, esta información se debe completar con mayor detalle. Por ello se exige un contrato de integración en el que se facilite toda la información relevante tanto de entrada como de salida para consumir el servicio y tratar adecuadamente las respuestas que éste proporcione, tanto por el sistema mediador (bus de interoperabilidad) como por el usuario/consumidor final.
- Dimensionamiento del Sistema. Los recursos a destinar a la hora de exponer los diferentes servicios, son dependientes en ocasiones, de las características como los perfiles de tráfico, tamaño de las peticiones/respuestas, etc. Es importante detallar el perfil de tráfico esperado en número de peticiones por minuto/segundo, nivel de concurrencia máximo, así como si el tráfico es estacionario, periódico, etc. Todas estas características, más las que se puedan considerar de interés por parte de los desarrolladores deben ser incorporadas en el formulario de alta de proveedor. Con ellas se definen los acuerdos de nivel de servicio a cumplir entre los diferentes componentes. Puede consultar los formularios de alta de proveedor en este enlace.
- Comunicaciones: En ocasiones, se producen restricciones de acceso administrativas, que obligan a considerar temas como direcciones IP origen y destino, protocolos a nivel de VPN, etc. La exposición de los servicios de la PAI se hace mediante protocolo https securizando las peticiones mediante WSSecurity en cabecera SOAP. No existe de momento exposición de servicios REST, JSON-RPC, etc.
- Requisitos de Integración: Ciertos componentes y servicios desplegados, con frecuencia requieren de la interacción con otros elementos o servicios que deban ser configurados y definidos en sincronía. Todas las interacciones y dependencias que sean necesarias para el correcto funcionamiento del servicio deben ser establecidas en la solicitud a la PAI mediante el formulario del proveedor, haciendo constar en la petición todos los requerimientos de integración accesorios que deban ser contemplados.

3.6 Servicios REST

Destacamos en este apartado por su creciente popularidad la arquitectura REST, definida como un conjunto de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. Se caracteriza por la siguiente serie de fundamentos clave:

- Un **protocolo cliente/servidor** sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.
- Un **conjunto de operaciones** bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son **POST, GET, PUT y DELETE**.
- Una **sintaxis universal** para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.
- El **uso de estándares**.

Si bien su popularidad, radica en su potencia, rendimiento y sencillez, en el ámbito que nos ocupa como es la PAI, no se contempla la inclusión de servicios REST ya que no soporta WS-Security, además SOAP proporciona una implementación estándar de integridad de datos y privacidad de datos. Así, este requerimiento de seguiridad, se estima necesaria en la plataforma.

4 Entorno tecnológico de la PAI

4.1 Política de seguridad

Las peticiones que se realizan a la PAI se securizan y autentican utilizando mecanismos WS-Security.

La PAI cuenta con una política de autorización de firma de mensaje en cabecera SOAP, siguiendo las especificaciones de WS-Security. A los servicios se les aplica una política que exige que las peticiones vengan firmadas mediante un certificado que pueda validar la ACCV.

Los consumidores de servicios de verificación de datos de la GVA o servicios instrumentales, pueden utilizar tanto certificado de Sello de Órgano como certificados de aplicación de la ACCV.

Los consumidores de servicios de verificación de datos ofrecidos por la Plataforma de Intermediación de Datos (PID) a través de la PAI, están obligados a utilizar certificados de Sello de Órgano de la ACCV para firmar sus peticiones.

Las particularización de los parámetros de configuración de seguridad se pueden ver en la siguiente figura:

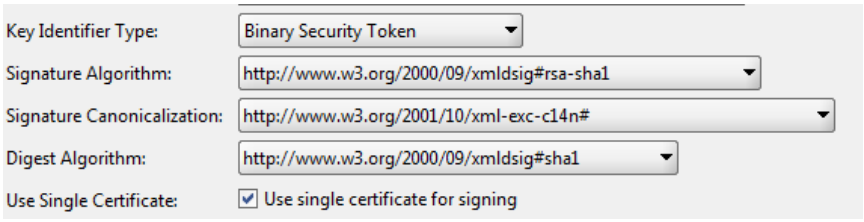


Ilustración 1: Definición de parámetros WSS

Especificación disponible en:
https://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf

4.2 Proveedores de servicios

Tal y como se describe en la parte de clientes de servicio, hay diversas tecnologías que hacen posible la generación de servicios web. En el 'Manual de Usuario para Generación de servicios en gvNIX' se muestran los pasos para la generación de un servicio web con el framework gvNIX siguiendo los patrones descritos en este documento.

En el desarrollo en sí de los servicios, existen en la actualidad multitud de entornos de desarrollo (frameworks) que permiten agilizar los procesos de integración, así como la publicación rápida de webservices centrándose únicamente en la lógica de negocio a implementar, obviando las labores accesorias. La facilidad e integración con los IDEs de uso

habitual (Eclipse, Netbeans, JDeveloper, IntelliJ, BlueJ, etc) permite un ágil y rápido desarrollo teniendo curvas de aprendizaje con tiempos muy cortos. En el punto 3 se esbozan directrices generales de orientación para el desarrollo de servicios en entornos SOA.

4.3 Trazabilidad en el consumo de servicios de la PAI

Las aplicaciones que consumen los servicios de la PAI, deben guardar información suficiente que permita hacer un seguimiento de la trazabilidad de las transacciones realizadas e intercambiadas con la plataforma.

Es importante destacar que las aplicaciones que hacen uso de la PAI, deben almacenar los correspondientes identificadores que permitirán hacer un seguimiento de dichas transacciones, ante futuras auditorias del proveedor del servicio.

Por parte de la PAI y de manera general a todos los servicios desplegados en la misma, se registran los siguientes parámetros de las peticiones realizadas;

- Cabecera de la petición (Header).
- Hash del cuerpo del mensaje (Body).
- Timestamp en el que se recibió la petición
- Servicio destino de la petición
- Tipo de mensaje; *request* | *response*
- Certificado del consumidor.
- Procedimiento de la petición.

Además específicamente para los ***servicios instrumentales*** se registra el “Id_trazabilidad” y para los ***servicios de verificación*** se registra el “IdPetición” y el/los “Idsolicitud”. Seguidamente procedemos a exponer estas peculiaridades y describimos las características que deben implementar las peticiones a realizar en las llamadas a los servicios expuestos en la PAI.

NOTA: Para los servicios con política aplicada se registra adicionalmente a todo lo descrito anteriormente el “thread-id” como identificador inequívoco del hilo.

4.3.1 Trazabilidad en los servicios instrumentales.

Con el fin de minimizar los cambios a realizar en los correspondientes desarrollos, e igualmente, impactar de la forma más liviana posible en el rendimiento de la plataforma, se ha optado por incorporar una cabecera de mensaje SOAP, en la petición al servicio web, que incluya el tag ‘Id_trazabilidad’ que, permitirá hacer el correspondiente seguimiento posterior.

Ejemplo:

```
<Id_trazabilidad xmlns="http://dgti.gva.es/interoperabilidad">68254ed222d65eeb-CODIGO_CATI-20140130184955000</Id_trazabilidad>
```

Este tag se compone del número de serie del certificado utilizado para firmar la petición, CODIGO_CATI el cual toma el valor del código CATI de la aplicación y el instante de la petición codificado como año,mes,dia,hora,minuto,segundo,milisegundos (patrón YYYYMMddHHmmssSSS). Estos tres componentes del tag de trazabilidad se han de separar por un carácter '-' como se aprecia en el formato de ejemplo mostrado anteriormente, se define además el namespace correspondiente "**http://dgti.gva.es/interoperabilidad**".

Un ejemplo de código en Java - CXF para incluir esta cabecera sería del estilo:

```
List<Header> headersList = new ArrayList<Header>();
Header testSoapHeader1 = new Header(new QName("http://dgti.gva.es/interoperabilidad", "Id_trazabilidad"),
  "68254ed222d65eeb-CODIGO_CATI-20140130184955000", new JAXBDataBinding(String.class));
headersList.add(testSoapHeader1);
client.getRequestContext().put(Header.HEADER_LIST, headersList);
```

Existen documentación de inclusión de esta misma cabecera referenciada arriba, en tecnología gvNIX (documento Manual_Usuario_Generación_servicios_gvNIX) y soapUI (documento GUIA_CONFIGURACION_SOAP_UI).

4.3.2 Trazabilidad en Servicios de Verificación

En el caso de los servicios de verificación, la trazabilidad ha de ser igualmente registrada por la aplicación consumidora del servicio, pero los valores a guardar, vienen directamente informados como parámetros/tags del mensaje. Para ello se registran siempre los campos de '*IdSolicitud*' (para peticiones con más de un "*IdSolicitud*" se registran todos los incluidos en el mensaje) y '*IdPeticion*' del mensaje (ambos son tags obligatorios String 16/26 caracteres).

Adicionalmente a estos dos campos, se ha de registrar el código del procedimiento administrativo que faculta y habilita la petición que se realiza por parte del organismo que cede los datos. Este procedimiento puede ser informado de dos formas. Por una parte, a través del tag '*CodProcedimiento*' que es informado directamente en el mensaje en gran parte de los servicios de verificación. Se debe poblar con el código GUC adecuado.

En los casos en los que este campo no sea informado, bien porque no exista, bien porque sea opcional, se deberá incluir el código de procedimiento, insertándolo en las primeras 20 posiciones del tag Finalidad, conforme al criterio que exponemos a continuación (el Id_expediente puede ir vacío).

Finalidad: **CodProcedimiento#::#Id_Expediente#::#Texto Finalidad**

4.4 Definición de Errores de negocio y SoapFaults

Es imprescindible contar con un tratamiento de errores definido y detallado para aportar la información necesaria por una parte al usuario o aplicación final consumidora, pero también para el entorno de interoperabilidad con el fin de que lo medie o derive al cauce adecuado.

Por ello, y siguiendo el mismo funcionamiento establecido en la Plataforma de Intermediación de Datos, se devolverá un mensaje SOAP Fault cuando el error detectado pertenezca a alguno de los siguientes tipos:

- Error de conexión a la BD.
- Error de conexión a los sistemas externos (SAFE, Servidores Externos, etc).
- Error en la validación de esquemas (o petición recibida sin firma).
- Error por Validación de la Firma digital
    - No estar firmada la petición
    - Certificado caducado, revocado o no válido
- Error del Sistema Interno en el tratamiento de la petición.

Los mensajes SoapFault no se firmarán.

4.4.1 Especificación Errores de negocio

En el resto de casos, no contemplados en la lista anterior, se entenderá que la petición se ha podido tramitar y se devolverá un mensaje de Respuesta especificando en las etiquetas correspondientes el código y el texto del error o estado correspondiente (una vez mapeado) al considerarse una respuesta contemplada por el negocio.

Esos códigos y texto deberán estar recogidos en el contrato de integración del servicio.

Un ejemplo de estos tipos de errores controlados por el servicio final y que deben ser tratados atendiendo a lo especificado en el contrato de servicio pueden ser a modo de ejemplo algunos como los que se expone en la siguiente tabla, en la que se recogen errores de negocio típicos de diferentes servicios estatales. El siguiente listado de códigos y descripciones de errores se devuelve en el nodo DatosEspecificos del esquema de respuesta, concretamente, en los campos CodigoEstado y LiteralError:

Código de respuesta	Descripción de la respuesta obtenida
0	El contribuyente se encuentra al corriente en sus obligaciones tributarias
1	El contribuyente no se encuentra al corriente en sus

	obligaciones tributarias.
2	Se ha encontrado más de un registro para los datos del titular indicados
3	Para la persona consultada no hay información en la Comunidad Autónoma.

Tabla 2.-Errores de negocio devueltos por servicio de estar al corriente de las obligaciones tributarias GVA

Para las peticiones que devuelven respuestas con errores de negocio, en el nodo “Estado” del nodo “Retorno" en “DatosEspecificos” del mensaje de Respuesta se devolverá información que indica que la petición es errónea.

4.4.2 Especificación Soap Fault

Según el esquema de SOAP, la forma del mensaje de la parte SOAP Fault es la siguiente:

```
<xs:element name="Fault" type="tns:Fault" />
<xs:complexType name="Fault" final="extension">
<xs:annotation>
        <xs:documentation>Fault reporting structure</xs:documentation>
</xs:annotation>
    <xs:sequence>
          <xs:element name="faultcode" type="xs:QName" />
          <xs:element name="faultstring" type="xs:string" />
          <xs:element name="faultactor" type="xs:anyURI" minOccurs="0" />
          <xs:element name="detail" type="tns:detail" minOccurs="0" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="detail">
    <xs:sequence>
        <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax" />
</xs:complexType>
```

Según acuerdo y acercándonos a los estándares los valores que tomarán cada uno de estos elementos será el siguiente:

4.4.2.1 faultcode

Se debería dejar abierta la especificación de este elemento, siempre y cuando sea un Xml Qualified Name correcto, según especificación de la W3C. Esto permitirá usar los faultcode

que se rellenan automáticamente por los motores SOAP. Opcionalmente, si se desea se podría indicar un faultcode del tipo:

Faultcode = **Sender | Client | Receiver | Server [.Subcode]***

SOAP 1.1	SOAP 1.2	Descripcion
Client	Sender	El *SOAPMessage* no se formó correctamente o no contiene la información necesaria para tener éxito.
Server	Receiver	El *SOAPMessage* no pudo ser procesado debido a un error de procesamiento.

Donde Sender o Client, se utilizarían para indicar que el problema proviene del mensaje enviado por el requirente, y Receiver o Server, para indicar que el problema ha surgido por los procesos del receptor del mensaje.

Si el mensaje de error se localizase en la PAI este campo vendría informado como "*Server*" o "Receiver" dependiendo de la versión de SOAP del servicio.

4.4.2.2 faultstring

Puede tomar cualquier valor de tipo string. El proveedor debe tratar este campo para devolver errores legibles. Nunca se debe devolver información sobre las IP's, URL u otros elementos sensibles para la seguridad del servicio.

Como sugerencia podría tomar el valor:

faultstring = [**Cod_Error**] **Literal Error**

Donde el Cod_Error serían 4 dígitos.

Por ejemplo: "[0301] Organismo no autorizado"

4.4.2.3 detail

Se recomienda que este elemento contenga una estructura/nodo Atributos, en la que se indicará toda la información necesaria para el error, con el fin de mantener una coherencia sintáctica con los errores proporcionados por los servicios expuestos en la Plataforma de Intermediación de Datos (PID).

Esquema de nodo Atributos:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://intermediacion.redsara.es/scsp/esquemas/V3/soap-faultatributos" xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
targetNamespace="http://intermediacion.redsara.es/scsp/esquemas/V3/soapfault-atributos" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="Atributos">
        <xs:complexType>
            <xs:all>
                <xs:element ref="IdPeticion"/>
                <xs:element ref="NumElementos"/>
                <xs:element ref="TimeStamp"/>
                <xs:element ref="Estado" minOccurs="0"/>
                <xs:element ref="CodigoCertificado"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:element name="CodigoCertificado">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:maxLength value="64"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="CodigoEstado">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:maxLength value="4"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="CodigoEstadoSecundario">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:maxLength value="16"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="IdPeticion">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:maxLength value="26"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="LiteralError">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:maxLength value="255"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="NumElementos">
        <xs:simpleType>
            <xs:restriction base="xs:int">
                <xs:totalDigits value="7"/>
                <xs:minInclusive value="0"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="TiempoEstimadoRespuesta">
        <xs:simpleType>
            <xs:restriction base="xs:int">
```

```
                    <xs:totalDigits value="4"/>
                    <xs:minInclusive value="0"/>
               </xs:restriction>
          </xs:simpleType>
     </xs:element>
     <xs:element name="TimeStamp">
          <xs:simpleType>
               <xs:restriction base="xs:string">
                    <xs:maxLength value="29"/>
               </xs:restriction>
          </xs:simpleType>
     </xs:element>
     <xs:element name="Estado">
          <xs:complexType>
               <xs:all>
                    <xs:element ref="CodigoEstado" minOccurs="0"/>
                    <xs:element ref="CodigoEstadoSecundario" minOccurs="0"/>
                    <xs:element ref="LiteralError" minOccurs="0"/>
                    <xs:element ref="TiempoEstimadoRespuesta" minOccurs="0"/>
               </xs:all>
          </xs:complexType>
     </xs:element>
</xs:schema>
```

4.4.2.4 Ejemplo de SoapFault

Un ejemplo de estos tipos de errores lo podemos ver en la siguiente tabla que es extraída de un contrato de integración.

Código de estado SCSP	Descripción del error	Cuando se devuelve este error
0252	Valor incorrecto para el campo <Nombre campo>	Se recibe una petición en la que el valor informado para un campo no es correcto
0301	Organismo no autorizado	El organismo que envía no está autorizado para consumir el servicio
0302	Certificado caducado	El certificado esta caducado
0303	Certificado revocado	El certificado esta revocado
0305	Firma no válida	La firma de la petición no es válida
0307	No se ha encontrado el nodo firma	La petición no tiene nodo de firma
0309	Error al verificar el certificado	Se produce un error al validar el certificado
0310	No se ha podido verificar la CA del certificado	Se produce un error al verificar la Autoridad de Certificación del certificado
0401	La estructura del XML introducido no corresponde con el esquema	La validación del esquema de la petición o de la solicitud de respuesta no resulta satisfactoria.
0402	Falta informar campo	Se recibe una petición en la que algún dato

	obligatorio <nombre_campo>	necesario para poder tramitar la petición no se ha enviado o se ha enviado vacío.
0901	Servicio no disponible	Imposible ejecutar el servicio.

Tabla 3.- Ejemplo de Errores devueltos como SOAPFault

Con lo expuesto anteriormente un ejemplo de SoapFault podría tener el siguiente aspecto:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
    <soapenv:Fault>
        <faultcode>env:Server</faultcode>
        <faultstring>[0403] Error al obtener el contenido XML del mensaje SOAP </faultstring>
        <faultactor>XSPRUW30 </faultactor>
        <detail>
        <Atributos xmlns="http://intermediacion.redsara.es/scsp/esquemas/V3/soapfaultatributos">
          <IdPeticion>CODCERT00001</IdPeticion>
          <NumElementos>1</NumElementos>
          <TimeStamp>2010-09-20T13:22:35.993+0200</TimeStamp>
          <Estado>
          <CodigoEstado>0403</CodigoEstado>
          <LiteralError>Imposible obtener el contenido XML del mensaje SOAP.</LiteralError>
          <CodigoEstadoSecundario>04031</CodigoEstadoSecundario>
          <LiteralErrorSec>Imposible obtener el campo atributos del SOAP</LiteralErrorSec>
          <TiempoEstimadoRespuesta>0</TiempoEstimadoRespuesta>
          </Estado>
          <CodigoCertificado>AEAT103I</CodigoCertificado>

        </Atributos>

        </detail>
    </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

5 Apoyo a los desarrolladores

La PAI pretende ofrecer herramientas e información al desarrollador, mediante la cual se pueda facilitar, reducir el tiempo y dedicación destinado a la integración con la Plataforma.

5.1 Esqueleto de servicio web

Definimos esqueleto de servicio como las clases necesarias para la creación de un servicio web según las directrices descritas en el documento que nos ocupa. Incluimos en este paquete; las clases "base" del servicio, así como las diseñadas para realizar otras acciones (seguridad, auditoria).

5.1.1 Como utilizar gvNIX en el ámbito de la PAI

gvNIX es un entorno de trabajo Java de código abierto para el desarrollo rápido de aplicaciones web, altamente productivo, flexible y que no compromete la calidad de los proyectos. Es una distribución de Spring Roo centrada en reutilizar el conocimiento y los estándares de las organizaciones para construir sus aplicaciones de gestión, cuyo organismo responsable es la Generalitat Valenciana.

Este entorno, dispone de Add-ons que proporcionan mediante la integración de CXF las herramientas necesarias para realizar fácilmente tanto clientes de servicios como servicios web.

La documentación y el propio entorno están disponibles en: http://www.gvnix.org/.
Toda la información referente a los pasos a seguir para realizar dicho esqueleto con tecnología gvNIX, se encuentran más detalladamente en el documento de "*e-SIRCA-CONSTRUCCION-Manual_Usuario_Generación_de_Servicios_gvNIX*".

5.1.2 Bloques que componen el esqueleto

Como hemos citado anteriormente el esqueleto se divide en varias partes, en adelante módulos.

Por un lado y más importante tenemos las clases necesarias para generar un servicio web a raíz de un WSDL y unos XML Schemas ya definido y válidos.

Por otro lado, tenemos los módulos destinados a auditoría, seguridad y gestión de errores. Describiendo un poco cada uno de estos últimos:

- Módulo auditoría: Se implementan las bases de trazabilidad y auditoria de los mensajes a la entrada y/o salida.
- Módulo ws-security: A través de este módulo, es posible tanto la firma de peticiones de respuesta mediante un certificado concreto y/o encriptación de parte del mensaje.

Además, es posible su extensión para el tratamiento de seguridad en la entrada del mensaje.

- Módulo gestión de errores: Mediante validaciones en la lógica del servicio se permite transmitir una respuesta errónea, que asociada a un esquema que se encargará de dar el detalle al error del servicio, cumpliendo con lo definido en 4.4.2 Especificación Soap Fault.

5.2 Clientes de servicios

Existen diversas tecnologías en la actualidad para consumir los servicios web. No es capítulo de este documento servir de guía tutorial de desarrollo, sino dar unas pautas de ejemplo. En el 'Manual de usuario para el consumo de servicios instrumentales y de verificación' se muestra un cliente de servicio realizado con Apache CXF 2.7. Concretando las recomendaciones que desde la PAI se han realizado, se optó por clientes basado en CXF 2.7 por su comodidad en la confección de los Stubs y clases asociadas al consumo de servicios web.