Oficina Técnica de Interoperabilidad

Normativa de Desarrollo de Servicios Web

Andalucía se mueve con Europa

Unión Europea

Fondo Europeo de Desarrollo Regional

JUNTA DE ANDALUCIA

Queda prohibido cualquier tipo de explotación y, en particular, la reproducción, distribución, comunicación pública y/o transformación, total o parcial, por cualquier medio, de este documento sin el previo consentimiento expreso y por escrito de la Junta de Andalucía.

HOJA DE CONTROL

Documento	Normativa de Desarrollo de Servicios Web	**Versión**	1.8
Proyecto	Oficina Técnica de Interoperabilidad		
Elaborado por	OTI	**Fecha**	04/02/2020
Validado por		**Fecha**	
Aprobado por		**Fecha**	
Fichero	OTI_INT-NORM_Normativa de Desarrollo de Servicios Web_v01r08.odt		
Plantilla	OAC-Formato_Horizontal-PLT	**Versión**	1.0

Control de Modificaciones

Versión	Fecha	Autor	Descripción del Cambio
1.0	23/04/2018	everis	Versión inicial.
1.1	18/06/2018	everis	- Se especifica la obligatoriedad de cumplimentar cada una de las plantillas. - Se describe que la información de los servicios siempre deberá estar disponible para todos los equipos implicados en el apartado Identificación y responsabilidad única del servicio web - Se añade el apartado Entregables suceptibles de entrega/revisión, donde se describe el ciclo de vida de los entregables y su grado de afección. - Se añade el apartado Conjunto de métricas de evaluación de un Servicio Web - Se añade el anexo Ejemplos de peticiones y respuestas SOAP
1.2	12/12/2018	OTI	Revisión del documento por parte del equipo de la OTI: - Se añade normativa general y específica de REST y SOAP para el desarrollo de servicios. - Se amplia la normativa de seguridad para incluir aspectos relativos a la seguridad en la entrada y salida de los servicios de la Plataforma de Interoperabilidad. - Se añade normativa de codificación de caracteres. - Se amplía la normativa de servicios REST para añadir aspectos relativos al content-type, operaciones, versionado y esquemas de mensajería. - Matices realizados en todo el documento para una mayor claridad y concisión de las normas a cumplir.
1.3	05/02/2019	OTI	Cambios realizados en el contenido de los apartados en base a las observaciones del consultor experto.
1.4	11/03/2019	OTI	Cambios realizados tras la validación de la Directora de Proyecto: - Apartado Recursos y artefactos corregido para aclarar que es una normativa que afecta a todos los recursos, no sólo a las secuencas. - Apartado Archivo WSDL corregido para que el namespace, en

			lugar del texto "cmaot" incluya el nuevo valor "medioAmbiente". - Corregidas descripciones de los ejemplos del Anexo II para evitar incongruencias.
1.5	15/03/2019	OTI	Ampliada la sección de entregables por servicio para incluir nuevos documentos defindos.
1.6	21/08/2019	OTI	Cambios realizados en los nombre de las plantillas referenciadas en el documento para su adecuación a lo publicado por la OTI. Eliminado el esquema genérico de error funcional en el apartado sobre errores funcionales SOAP, debido a que vendrán definidos en cada uno de los contratos de integración de cada servicio. Se eliminan el apartado ***"Modelo de seguridad para los Servicios de Interoperabilidad"*** ya que en breve se publicará una normativa enfocada a seguridad.
1.7	17/10/2019	OTI	Cambios realizados en el texto y en el diagrama de la estructura de las componentes del Servicio Web. Cambio en el diagrama del apartado de control de versiones del código.
1.8	04/02/2020	OTI	Se cambia la nomenclatura que deben seguir los recursos dentro del proyecto CP y se incluyen ejemplos para secuencias, message-procesors y message-store

Índice

1. INTRODUCCIÓN....8
1.1. Objeto y alcance del documento....8
2. NORMAS Y RECOMENDACIONES PARA EL DESARROLLO DE SERVICIOS WEB DE LA CONSEJERÍA....9
2.1. Normas comunes....12
2.1.1. Identificación y responsabilidad única del servicio web....12
2.1.2. Perfil de Integración....12
2.1.3. Contrato del Servicio Web....13
2.1.3.1. Elementos comunes de los Contratos de Servicios Web....13
2.1.4. Entregables susceptibles de entrega/revisión....14
2.1.5. Política común de gestión de errores....15
2.1.5.1. Errores de comunicación....15
2.1.5.2. Errores de mensajería....15
2.1.5.3. Catálogo de errores funcionales....16
2.1.5.4. Servicio de notificación de error y recepción....16
2.1.6. Conjunto de métricas de evaluación de un Servicio Web....17
2.1.7. Desarrollo de servicios....18
2.1.7.1. Nomenclatura y estructura de los componentes del Servicio Web....18
2.1.7.2. Propiedades de proyectos Maven....19
2.1.7.3. Recursos y artefactos....19
2.1.7.4. Control de versiones del código....21
2.1.7.5. Log....22
2.1.7.6. Timeout....23
2.1.7.7. Política de reintentos....24
2.1.7.8. Aseguramiento del mensaje....25
2.1.7.9. Uso de librerías externas....26
2.1.7.10. Codificación de caracteres....26
2.2. Normas para servicios tipo SOAP....27
2.2.1. Archivo WSDL....27
2.2.2. Definición de servicios....27
2.2.3. Patrón de diseño XML Schema....28
2.2.4. Definición de atributos y operaciones....28
2.2.5. Intercambio de documentos a través de los servicios....28
2.2.6. Versionado de servicios y de esquemas de datos....29
2.2.6.1. Definiciones y renombrado de versiones....29
2.2.6.2. Adaptación de los clientes....29
2.2.7. Versionado de especificación....29
2.2.8. Equilibrio entre servicios y operaciones....29
2.2.9. Content-Type....30
2.2.10. Gestión de errores en servicios web SOAP....30
2.2.11. Desarrollo de servicios SOAP....30
2.2.11.1. Recursos y artefactos....30
2.3. Normas para servicios tipo REST....32
2.3.1. Uso de los métodos HTTP....32
2.3.2. Nomenclatura de los URIs....32
2.3.3. Versionado....34
2.3.4. JSON Schema....35
2.3.5. Content-Type....35
2.3.6. Gestión de errores en servicios web REST....35
2.3.7. Documentación de Servicios REST....36

2.3.8. Desarrollo de servicios REST....36
2.3.8.1. Recursos y artefactos....36
3. ANEXO I. Ejemplos de versionados de servicios web....37
3.1. Versión menor....37
3.2. Versión mayor....37
4. ANEXO II. Ejemplos de peticiones y respuestas SOAP....39

Índice de tablas

Tabla 1: Ficha resumen de normas de aplicación a los Servicios Web de la Consejería....11
Tabla 2: Aspectos del entregable Perfil de Integración....12
Tabla 3: Aspectos del entregable Contrato del Servicio Web....13
Tabla 4: Aspectos del entregable Elementos Comunes del Contrato del Servicio Web....14
Tabla 5: Fases de los entregables susceptibles de entrega/revisión....14
Tabla 6: Valores óptimos para las métricas de un Servicio Web....17
Tabla 7: Formato y descripción de los campos del URN en servicios web SOAP de la Consejería....27

Índice de ilustraciones

Figura 1: Versionado: Inclusión de un atributo no obligatorio a los datos de entrada (Versión menor)....37
Figura 2: Versionado: Añadir un nuevo método al Servicio(Versión menor)....37
Figura 3: Versionado: Eliminación de un método (Versión mayor)....38
Figura 4: Versionado: Renombrado de un método (Versión mayor)....38
Figura 5: Versionado: Cambio en los parámetros de un método (Versión mayor)....38
Figura 6: Versionado: Cambios en un tipo de datos existente dentro del esquema (Versión mayor)....38
Figura 7: Versionado: Transformar un campo opcional a obligatorio (Versión mayor)....38
Figura 8: Petición de ejemplo de un servicio web SOAP sin seguridad....39
Figura 9: Petición de ejemplo de un servicio web SOAP con seguridad UsernameToken....39
Figura 10: Respuesta de ejemplo de un servicio web SOAP sin errores....40
Figura 11: Respuesta de ejemplo de un servicio web SOAP con error SoapFault....40
Figura 12: Respuesta de ejemplo de un servicio web SOAP con error funcional....40

1. INTRODUCCIÓN

1.1. Objeto y alcance del documento

El presente documento se enmarca en el proyecto Oficina Técnica de Interoperabilidad y tiene como objeto establecer la normativa a seguir sobre la definición e implementación de Servicios Web que se desarrollen en la Consejería.

Estas normas son de obligado cumplimiento, debiendo valorarse aquellos casos excepcionales. Si algún actor considera necesario el incumplimiento total o parcial de alguna de las normas y/o recomendaciones contenidas en este documento tendrá que documentar de forma fehaciente su justificación, así como aquella documentación adicional que se requiera para su validación por parte de la Dirección Técnica de la Consejería. Del mismo modo, habrá de documentarse la solución adoptada que tendrá que intentar adecuarse a las normas contempladas en esta política.

Este procedimiento va dirigido a todo el personal con responsabilidades en la toma de decisiones dentro de cada ámbito de actuación que se defina en el documento, tales como:

- **Directores de proyectos**, que han de velar por el conocimiento y puesta en práctica de la normativa por parte de los equipos de desarrollo.
- **Equipo de desarrollo**, que ha de cumplir las normas aquí presentadas.
- **Equipo de calidad**, que ha de velar por el cumplimiento de las normas aquí presentadas.
- **Equipo de explotación**, que ha de gestionar y administrar los componentes software objeto de este documento.
- **Equipo de Interoperabilidad**, responsable de la Plataforma y del gobierno de la interoperabilidad.

En concreto, se desarrollan en este documento los siguientes apartados:

- **Normas comunes.** Normas y recomendaciones de aplicación al conjunto de servicios web, tales como normas acerca de la identificación de los servicios web, elaboración de los perfiles de integración, Contrato del Servicio Web y la política común de gestión de errores de los servicios web.
- **Normas para servicios tipo SOAP**. Especificaciones y directrices para los servicios web con arquitectura SOAP acerca de la implementación del archivo WSDL, la definición de este tipo de servicios, la aplicación de un patrón de diseño XML Schema, exposición de operaciones, intercambio de documentos, nomenclatura a seguir, criterios para el versionado del servicio, gestión de los errores, etc.
- **Normas para servicios tipo REST**. Normas y recomendaciones para los servicios de este tipo de arquitectura, entre otras para el uso de los métodos HTTP, la nomenclatura en los URIs, versionado de los servicios y gestión de errores.
- **Modelo de seguridad para los Servicios de Interoperabilidad.** Recomendaciones acerca de las medidas y mecanismos de seguridad generales y según la red desde la que sean accesibles.

2. NORMAS Y RECOMENDACIONES PARA EL DESARROLLO DE SERVICIOS WEB DE LA CONSEJERÍA

A continuación se recogen las normas y recomendaciones que han de seguirse en el desarrollo y puesta a disposición de los servicios web provistos por los sistemas de información de la Consejería.

En la siguiente tabla **se recogen de manera general las normas** contenidas en el presente documento:

APARTADO	DENOMINACIÓN	DESCRIPCIÓN
2.1	**NORMAS COMUNES**	Normas que se han de aplicar a todos los servicios web de la Consejería, indistintamente del tipo de arquitectura.
2.1.1	**Identificación y responsabilidad única del servicio web**	Directrices acerca de la obligación de diferenciar cada servicio web conforme a los servicios de negocio a los que proporcione soporte.
2.1.2	**Perfil de Integración**	Directrices sobre cómo elaborar el Perfil de Integración para definir la interacción entre varios servicios.
2.1.3	**Contrato del servicio web / Elementos comunes**	Establece que cada servicio web ha de contar con su correspondiente Contrato de Servicio y qué información ha de contener. Del mismo modo, determina qué elementos comunes de los servicios web de cada ámbito han de ser documentados de manera diferenciada. De este modo, aquellos elementos (mensajes, atributos, datos paramétricos...) que se utilicen en más de un contrato de dicho ámbito, podrán ser referenciados, contemplándose exclusivamente en un documento.
2.1.4	**Documentos susceptibles de entrega/revisión**	Se definen los entregables obligatorios a elaborar y entregar en las fases de construcción y evolutivo/revisión
2.1.5	**Política común de gestión de errores**	Se indica el tratamiento que habrá que realizarse tanto con errores de carácter técnico, desacoplados del servicio y de la arquitectura del sistema, como aquellos errores funcionales derivados del propio tratamiento del mensaje.
2.1.6	**Conjunto de métricas de evaluación de un Servicio Web**	Se definen las métricas necesarias a cumplimentar en la definición del Contrato del Servicio Web
2.1.7	**Desarrollo de servicios**	Normas generales para que el desarrollo de los servicios, sus propiedades y los artefactos publicados en la Plataforma de Interoperabilidad sean abordados siempre de una forma homogénea.
2.2	**NORMAS PARA SERVICIOS TIPO SOAP**	Normas cuya aplicación se circunscribe a los servicios web de la Consejería que tengan una arquitectura SOAP.
2.2.1	**Archivo WSDL**	Fija que cada servicio web tipo SOAP ha de contar con el

APARTADO	DENOMINACIÓN	DESCRIPCIÓN
		correspondiente archivo WSDL publicado, así como las directrices generales que ha de cumplir dicho archivo. También se establece cómo han de ser los *namespaces* que se utilicen.
2.2.2	**Definición de servicios**	Se establece que ha de seguirse una aproximación *"Contract-first"* para el desarrollo de los servicios web SOAP.
2.2.3	**Patrón de diseño XML Schema**	Se definen las directrices en el diseño de los esquemas XML / XSD.
2.2.4	**Definición de atributos y operaciones**	Restricciones en los parámetros de operaciones expuestas por el servicio web.
2.2.5	**Intercambio de documentos**	Directrices sobre el intercambio de documentos a través de los servicios web.
2.2.6	**Versionado de servicios y de esquemas de datos**	Establece los criterios para la diferenciación y denominación de las versiones de los servicios web a lo largo de su ciclo de vida, así como para la adaptación de los clientes consumidores de los servicios web a los cambios que conlleven.
2.2.7	**Versionado de especificación**	Obligación de desarrollar los servicios web empleando misma versión de especificación o compatible.
2.2.8	**Equilibrio entre servicios y operaciones**	Recomendación para equilibrar el número de servicios y operaciones disponibles en cada uno de ellos.
2.2.9	**Content-Type**	Establece los valores que han de cofigurarse en el campo Content-Type de la cabecera del mensaje.
2.2.10	**Gestión de errores en servicios SOAP**	Especifica cómo debe implementarse la respuesta de servicios SOAP ante los errores que se produzcan.
2.2.11	**Desarrollo de Servicios SOAP**	Normas específicas para el desarrollo de servicios SOAP en la Plataforma de Interoperabilidad.
2.3	**NORMAS PARA SERVICIOS TIPO REST**	Normas cuya aplicación se circunscribe a los servicios web de la Consejería que tengan una arquitectura REST.
2.3.1	**Uso de los métodos HTTP**	Directrices para la utilización de los principales métodos HTTP en los servicios web.
2.3.2	**Nomenclatura URIs**	Reglas que han de seguir las URIs (Uniform Resources Identifier).
2.3.3	**Versionado**	Especificaciones para diferenciar las versiones del servicio conforme a sus actualizaciones o modificaciones.
2.3.4	**JSON Schema**	Normativa sobre el uso de esquemas en mensajería en formato JSON.
2.3.5	**Content-Type**	Establece los valores que han de configurarse en el campo Content-

APARTADO	DENOMINACIÓN	DESCRIPCIÓN
		Type de la cabecera del mensaje.
2.3.6	**Gestión de errores en los servicios REST**	Especifica la respuesta de los servicios REST ante los errores.
2.3.7	**Documentación de Servicios REST**	Normativa relativa a la documentación de la definición de los servicios REST en la Consejería.
2.3.8	**Desarrollo de servicios REST**	Normas específicas para el desarrollo de servicios REST en la Plataforma de Interoperabilidad.

Tabla 1: Ficha resumen de normas de aplicación a los Servicios Web de la Consejería

2.1. Normas comunes

2.1.1. Identificación y responsabilidad única del servicio web

Cada servicio web ha de tener una responsabilidad única y diferenciada que permita la clara identificación de los servicios de negocio.

Para ello, las operaciones que ofrezca el servicio web han de ser coherentes con la parte del negocio que representan, no permitiéndose la encapsulación en un solo servicio de toda la funcionalidad aplicable a un determinado modelo de negocio complejo.

Los servicios web que se diseñen tendrán responsabilidades bien repartidas, de tal manera que sean escalables y reutilizables.

La información de los servicios, como por ejemplo el catálogo de servicios de la Consejería, deberá estar disponible en un lugar accesible para todos los equipos, proporcionando la posibilidad de reutilizar los servicios ya definidos en el catálogo.

2.1.2. Perfil de Integración

Cada vez que exista una necesidad de integración con varios servicios, habrá que definir la interacción entre ellos mediante un *perfil de integración.*

El *Perfil de Integración* se recogerá en un documento que explique la solución de integración diseñada para una necesidad concreta. La plantilla que seguirá este documento es ***OTI_INT_Plantilla Perfil de Integración***, en su última versión, siendo obligatoria la cumplimentación de todos sus aspectos.

El Perfil de Integración recogerá los siguientes aspectos:

DENOMINACIÓN	DESCRIPCIÓN
Objetivos del proceso de interoperabilidad	Explicación de la necesidad de integración
Alcance del proceso de interoperabilidad	Funcionalidades, contexto y requisitos
Descripción general	Especificación de los actores y casos de uso que se darán en la integración junto con el flujo de ejecución de cada uno de ellos
Definición dinámica	Para cada uno de los casos de usos expuestos en el punto anterior debe definirse un diagrama de secuencia
Definición estática	Se especificarán cada uno de los servicios nombrados en el documento. Para cada servicio se informará sobre nombre, operación, proveedor, tipología del servicio y *contrato del servicio*

Tabla 2: Aspectos del entregable Perfil de Integración

Adicionalmente se podrán añadir aquellos aspectos que se entiendan oportunos para describir de forma correcta el perfil de integración.

2.1.3. Contrato del Servicio Web

Cada Servicio Web desarrollado ha de disponer de un Contrato del Servicio asociado. Este contrato seguirá la plantilla ***OTI_INT_Plantilla Contrato Servicio Web***, en su última versión, siendo obligatoria la cumplimentación de todos sus aspectos detallados a continuación:

DENOMINACIÓN	DESCRIPCIÓN
Servicio	En este apartado se describe los principales aspectos del servicio: nombre, tipología, objetivo, patrón del endpoint, versión, seguridad, autenticación, namespace, descripción del servicio web, alertas SLA, tiempos de respuesta y otras observaciones que se consideren necesarias
Operaciones	Para cada una de las operaciones del servicio se describe: nombre, objetivo, método HTTP (si procede), encoding, mensaje de entrada y salida, junto con sus respectivos esquemas y parámetros de entrada/salida. En el caso de servicios REST, se considera una operación diferente cada una de las operaciones HTTP que permite realizar este protocolo (GET, POST, PUT, DELETE).
Tipos de Datos	Descripción de los tipos de datos específicos del servicio que dan soporte a las operaciones del mismo
Tablas de referencia	Descripción de la información, específica del servicio, que por razones de diseño se intercambie en el servicio mediante su código correspondiente
Tablas de referencia comunes	Especificar que aquellas tablas de referencias comunes para más de un servicio dentro del mismo ámbito de negocio vendrán informados en el documento de elementos comunes, cuya plantilla es ***OTI_INT_Plantilla Elementos Comunes***, en su última versión.
Gestión de errores	Para cada uno de los errores que puede devolver el servicio, se define el código, la descripción del error y la recomendación para subsanar el mismo
Ejemplos	Ejemplos de todas las funcionalidades que contienen las operaciones del servicio
Información complementaria	Anexos, glosario y referencias

Tabla 3: Aspectos del entregable Contrato del Servicio Web

2.1.3.1. Elementos comunes de los Contratos de Servicios Web

Para favorecer la facilidad de mantenimiento de la documentación relativa a los Contratos de los Servicios Web, se recogerá en un documento diferenciado para cada sistema de información los elementos comunes a todos los servicios web que provea.

De esta forma, en el contrato del servicio aparecerá únicamente la información especifica del servicio, mientras que aquellos aspectos que sean comunes para los distintos servicios Web podrá consultarse en el documento de Elementos Comunes de los Contratos de Servicios Web del Sistema (plantilla ***OTI_INT_Plantilla Elementos Comunes***, en su última versión, siendo obligatoria la cumplimentación de todos sus aspectos), que contendrá los siguientes apartados:

DENOMINACIÓN	DESCRIPCIÓN
Tipos de datos	Recoge aquellos datos estructurados comunes a varios servicios Web del Sistema en cuestión
Catálogos maestros	Recoge los catálogos maestros del sistema que son compartidas por varios servicios web

DENOMINACIÓN	DESCRIPCIÓN
Gestión errores	Muestra los errores que pueden darse de manera genérica en varios servicios del Sistema

Tabla 4: Aspectos del entregable Elementos Comunes del Contrato del Servicio Web

2.1.4. Entregables susceptibles de entrega/revisión

La siguiente matriz muestra los entregables susceptibles de entrega/modificación en las diferentes fases de un proyecto de integración:

	FASES	
ENTREGABLES	**Construcción**	**Evolutivo/Revisión**
Perfil de Integración	OBLIGATORIO. Deberá haber sido elaborado y entregado.	CONDICIONAL. Solo será susceptible de modificación si se han producido cambios en alguno de los aspectos que se detallan.
Contrato del Servicio Web	OBLIGATORIO. Deberá haber sido elaborado y entregado	CONDICIONAL. Es obligatorio realizar la modificación en el caso de que se cambie la estructura de los mensajes, actualizándose los ejemplos aunque se traten de cambios en elementos comunes, o cualquier detalle específico del servicio. De este modo el cliente no se verá afectado y puede actualizar su interfaz web utilizando el nuevo contrato.
Elementos comunes del Contrato del Servicio Web	CONDICIONAL. Deberá haber sido elaborado y entregado, siempre que pueda ser aplicada la reutilización de los elementos definidos	CONDICIONAL. Solo será susceptible de modificación si alguno de los elementos comunes definido se ve modificado.
Análisis SOA	CONDICIONAL. En caso de haberse generado, será obligatorio.	CONDICIONAL. En caso de haberse generado, será obligatorio.
Plan de Pruebas Funcionales	OBLIGATORIO. Deberá haber sido elaborado y entregado.	OBLIGATORIO. Deberá haber sido elaborado y entregado.
Plan de Pruebas de Integración	OBLIGATORIO. Deberá haber sido elaborado y entregado.	OBLIGATORIO. Deberá haber sido elaborado y entregado.
Manual de Entrega	OBLIGATORIO. Deberá haber sido elaborado y entregado.	OBLIGATORIO. Deberá haber sido elaborado y entregado.

Tabla 5: Fases de los entregables susceptibles de entrega/revisión

El ciclo de vida que se deberá seguir para la entrega es:

1. Análisis SOA.
2. Perfil de Integración.
3. Contrato del Servicio Web.
4. Elementos comunes del Contrato del Servicio Web. La elaboración de este entregable solo aplicará en función de la reutilización de los elementos.
5. Plan de Pruebas de Integración.
6. Plan de Pruebas Funcionales.
7. Manual de Entrega.

La cumplimentación, validación y entrega de esta documentación definida en el ámbito de la interoperabilidad, debe comenzar durante la toma de requisitos y debe estar finalizada antes de comenzar la fase de construcción.

En evolutivos o revisiones del software en un proceso de integración, el Contrato del Servicio Web será el documento más importante a actualizar, ya que propiciará que los nuevos clientes, o incluso los ya existentes, se percaten de los nuevos cambios producidos en las interfaces web. Los documentos del Perfil de Integración y Elementos comunes, solo será necesaria su actualización si se modifica algún aspecto de ellos, por ejemplo si los requisitos de integración han sufrido modificaciones o si algunos elementos comunes, utilizados en alguna interfaz, han sido modificados.

2.1.5. Política común de gestión de errores

Se definen normas para el tratamiento de errores de comunicación, técnicos de aplicación y funcionales de aplicación.

2.1.5.1. Errores de comunicación

Se trata de aquellos relativos al envío y aceptación de mensajes por parte del sistema receptor de la misma.

Los receptores de la mensajería deberán realizar una validación sintáctica de:

- Esquema del mensaje
- Valores de los campos del mensaje (opcionalidad y/o cardinalidad)
- Formato de los valores de los campos (cuando se encuentre definido)

Si el mensaje no supera las validaciones se deberá informar al sistema origen de la llamada.

Debe tenerse en cuenta, de cara a la validación del esquema del mensaje, que los servicios pueden evolucionar, pero siempre deben ser retrocompatibles, es decir, se podrán añadir nuevos campos opcionales, los cuales, tanto si son enviados como si no, serán válidos para todas las versiones del servicio. Es por esto que los campos opcionales nunca deberán ser validados o, en caso incorporarse algunos nuevos en versiones futuras del servicio, se producirán errores ante mensajes válidos.

2.1.5.2. Errores de mensajería

1. <u>Gestión de los errores en la entrega del mensaje.</u>
    - Los consumidores del servicio web han de definir mecanismos para proceder cuando reciben un error en la entrega del mensaje.
    - La recomendación es que sólo se dé por finalizada la operación o recurso cuando se haya entregado. Para ello, todos los servicios, independientemente de que estén definidos como síncronos o asíncronos, deben tener una respuesta para verificar que el mensaje se ha entregado correctamente y que además cumple con unas validaciones mínimas. En el caso de que se produzca algún problema en la comunicación, el propio protocolo de comunicación informará del error al origen de la emisión.
2. <u>Gestión de los errores en la validación de un mensaje recibido</u>
    - Una vez que el mensaje es recibido por el sistema proveedor debe realizarse una validación sintáctica del mensaje. Esta validación se realizará a partir del esquema del mensaje, XSD o JSON Schema, fichero que debe ser conocido por el sistema consumidor, junto con el contrato del servicio, y que cubrirá esquema, valores de los campos (opcionalidad y/o cardinalidad) y formato.
    - En servicios con funcionamiento asíncrono, cuando un mensaje supera la validación a realizar, el sistema proveedor debe generar un mensaje informando que la comunicación ha sido correcta. Por el contrario, si se produce algún problema en la validación, el sistema proveedor deberá generar un mensaje informando del error producido al sistema origen.
    - En servicios con funcionamiento síncrono, en el caso en el que se produzca un problema de validación, el sistema proveedor deberá generar un mensaje informando del error producido al sistema origen y detener la ejecución del

proceso. Por el contrario, si la validación del mensaje se supera sin problema, no se generará una respuesta indicando que el mensaje es correcto, sino que se continuará con el proceso síncrono y se devolverá una respuesta con el contenido que corresponda en función del servicio.

3. Errores en el procesamiento del mensaje recibido. Cuando el procesamiento desconectado de un mensaje genere o detecte un error funcional o semántico, caben dos posibilidades:

   a) Error interno en la aplicación destino. Una vez entregado el mensaje, es responsabilidad del sistema receptor tratar el mensaje recibido. Durante el tratamiento del mensaje pueden darse errores del tipo tabla bloqueada, caída de BBDD, errores de ejecución, etc., que deben ser informados al sistema que invocó el servicio. En función del tipo de servicio, el comportamiento deberá ser el siguiente:

      - **Servicios síncronos**: la respuesta del mensaje llevará información del error producido.
      - **Servicios asíncronos**: se informará al sistema origen del error producido a través de un servicio asíncrono de notificación de error. Para ello, el sistema origen debe tener implementado el servicio para poder recibir dicha notificación.

   b) Error provocado por el contenido del mensaje. Estos errores deben ser reportados al origen, se tratan básicamente de errores causados por información incorrecta contenida en el mensaje y que pueden provocar inconsistencia de datos entre los sistemas integrantes. Existen dos situaciones:

      - **Servicios síncronos**: la respuesta del mensaje llevará información del error producido.
      - **Servicios asíncronos**: se informará al sistema origen del error producido a través de un servicio asíncrono de notificación de error. Para ello, el sistema origen debe tener implementado el cliente del servicio para poder recibir dicho notificación.

2.1.5.3. Catálogo de errores funcionales

Para conocer los errores funcionales que pueden presentarse en cada servicio debe consultarse el Contrato de Servicio correspondiente, donde se expondrán aquellos errores que sean propios del servicio. También, puede producirse algún error funcional que sea genérico para varios servicios existentes en el sistema de información proveedor. Para consultar estos últimos es necesario dirigirse al documento de elementos comunes de los Contratos de Servicios del sistema correspondiente.

2.1.5.4. Servicio de notificación de error y recepción

En el supuesto de que el mensaje del servicio sea asíncrono, si estuviera disponible, el sistema proveedor debe implementar la invocación a un servicio que permita informar al sistema origen del error producido, y por consiguiente, el sistema origen debe implementar un servicio para la recepción de dicha notificación.

De este modo, el sistema origen de la mensajería podrá disponer de una trazabilidad completa de los mensajes que ha emitido, pudiendo realizar correcciones o gestiones posteriores de los mismos en el caso de que se hayan detectados problemas en el sistema receptor.

2.1.6. Conjunto de métricas de evaluación de un Servicio Web

MÉTRICA	ENTORNO	VALOR ÓPTIMO	CRITICIDAD	DESCRIPCIÓN
Tasa de error	**Pruebas**	**5%**	**Media**	Mide en % el número de errores producidos en la conexión con el servicio sobre el total de peticiones.
	Producción	**0% en servicios impacto alto en la Consejería. En otros servicios es admisible una tasa de error inferior al 1%.** **En cualquier caso se ha de garantizar la entrada del mensaje**	**Alta**	
Tiempo medio de respuesta	**Pruebas**	**0 – 5 segundos**	**Media**	Tiempo medio de respuesta entre todas las peticiones del servicio en un tiempo determinado (Mínimo 1 hora)
	Producción	**0 – 2 segundos**	**Alta**	
Tiempo máximo de respuesta	**Pruebas**	**10 segundos**	**Media**	Tiempo máximo de respuesta del servicio (Timeout)
	Producción	**5 segundos**	**Alta**	
Escalabilidad	**Pruebas**	**SI**	**Baja**	Capacidad de escalado horizontal del sistema proveedor (SI o NO)
	Producción	**SI**	**Alta en servicios de impacto alto en la Consejería.** **Media en el resto.**	
Grado de concurrencia	**Pruebas**	**Mínimo 20**	**Media**	Peticiones por segundo que el servicio es capaz de procesar.
	Producción	**Mínimo 50**	**Alta**	
Tiempo de reintento (servicios asíncronos)	**Pruebas**	**30 segundos**	**Media**	Tiempo que debe pasar entre dos intentos de envío de un mensaje en caso de destino no disponible. Variará en Producción en base a volumetría, criticidad, rendimiento del servicio, etc.
	Producción	**10 – 30 segundos**	**Alta**	
Tiempo máximo de reintento (servicios asíncronos)	**Pruebas**	**1 día**	**Media**	Tiempo mínimo en el que un mensaje asíncrono encolado tendrá que mantenerse y no ser descartado. Variará en Producción en base a la volumetría, criticidad, rendimiento del servicio, etc.
	Producción	**3 – 5 días**	**Alta**	

Tabla 6: Valores óptimos para las métricas de un Servicio Web

Los SLAs incluidos en la plantilla deben ser verificables y medibles. Valores que se salgan de los óptimos indicados tendrán que ser debidamente justificados.

2.1.7. Desarrollo de servicios

A continuación se abordan todos aquellos puntos que deben ser tenidos en cuenta para que el desarrollo de los servicios, artefactos y propiedades publicados en la Plataforma de Interoperabilidad sigan los mismos criterios de nomenclatura, estructura y arquitectura.

2.1.7.1. Nomenclatura y estructura de los componentes del Servicio Web

Es fundamental que todos los Servicios Web publicados en la Plataforma de Interoperabilidad mantengan una estructura similar y que su nomenclatura sirva para identificar tanto al servicio como a cada uno de sus componentes. Es por eso que en este apartado se describen las pautas a seguir a la hora de crear nuevos servicios en la herramienta Developer Studio de WSO2.

La nomenclatura de los proyectos que se crearán para su publicación en la Plataforma de Interoperabilidad por cada servicio es la siguiente:

- **Maven Multi Module Project: [NOMBRE_DEL_SERVICIO]Parent**

  Este proyecto englobará todos los módulos/proyectos (CP, RP, CApp) que componen un config project service, ya que por defecto el Developer Studio te crea los proyectos por separado con sus propios *pom.xml*. Gracias a la creación del proyecto Maven Multi Module Project podremos englobar todos ellos en un solo proyecto, mejorando los aspectos de visualización del código en el Developer Studio.

  Solo debe existir un proyecto de este tipo por cada servicio.

- **ESB Config Proyect: [NOMBRE_DEL_SERVICIO]_CP**

  Este proyecto contendrá todos los artefactos que no vayan a ser almacenados como recursos del repositorio. Los artefactos que principalmente se almacenarán en este proyecto son: servicios proxy, secuencias y plantillas.

  Solo debe existir un proyecto de este tipo por cada servicio.

- **Registry Resources Project: [NOMBRE_DEL_SERVICIO]_RP**

  Este proyecto contendrá todos los artefactos que van a ser almacenados como recursos del repositorio, en concreto en la rama de gobernanza (*governance)*. Los artefactos que principalmente se almacenarán en este proyecto son: endpoints, ficheros de parámetros, scripts, transformadores XSL, políticas y WSDLs.

  Solo debe existir un proyecto de este tipo por cada servicio.

- **Data Service Project: [NOMBRE_DEL_SERVICIO]_DSS**

  Este proyecto contendrá todos los artefactos cuya funcionalidad es el acceso seguro a los datos alojados en cualquier RDBMS, CSV, Excel, ODS, Cassandra, Google Spreadsheets o RDF, y gestionar la transformación y validación de los datos a diferentes formatos. Te permite acceder a diferentes almacenamientos con peticiones SOAP.

  Solo debe existir un proyecto de este tipo por cada servicio.

- **Composite Application Project: [NOMBRE_DEL_SERVICIO]_CApp**

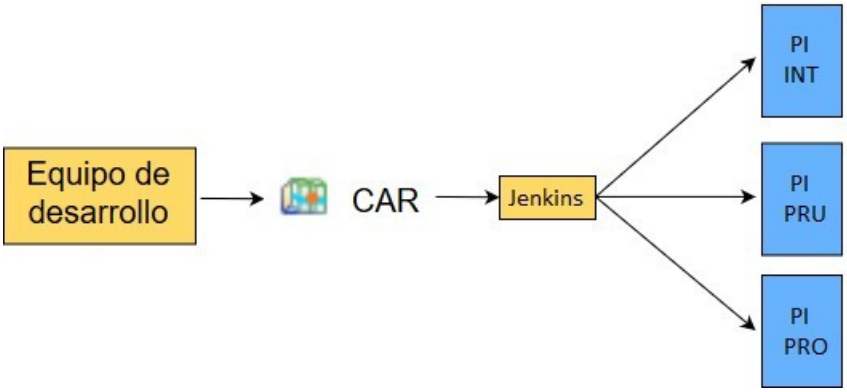
  Este proyecto será el encargado de generar el CAR (Composite Application Archive) donde se empaquetarán todos los artefactos y recursos del servicio que deseemos desplegar en el bus. Los artefactos y recursos a desplegar, vendrán identificados en el fichero pom.xml, teniendo siempre en cuenta que el rol debe ser *EnterpriseServiceBus.*

Para definir los nombres de todos los proyectos del Servicio Web se debe tener en cuenta lo siguiente:

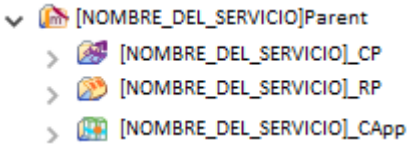
- **[NOMBRE_DEL_SERVICIO]** será el nombre del servicio según el estilo de escritura CamelCase con la primera letra en mayúsculas. Por ejemplo: ServicioDePrueba.

- Debe describir lo mejor posible la funcionalidad. De tal forma que se facilite a los desarrolladores y administradores de sistemas identificar la función del servicio basándose en su nombre.
- Hay que tener en cuenta que se distingue entre mayúsculas y minúsculas.

Teniendo todo en cuenta, de cara el despliegue de un servicio, siempre se deberá entregar a la OTI, un CAR que será desplegado desde Jenkins.



Es por eso que la estructura de proyectos que debe seguir cualquier servicio, como mínimo, es la siguiente:



2.1.7.2. Propiedades de proyectos Maven

Todos los proyectos que se pueden crear desde el Developer Studio, siguen una estructura de proyecto Maven, por ello cada vez que se crea un proyecto, independiente de la índole que sea, siempre se solicitará que se indique un groupId, artifactId y version. Se deberán definir siguiendo la guía de convención de nombres de Maven:

- **groupId**: identificará de forma única al proyecto frente a todos los demás proyectos. Deberá incluir el nombre del dominio y a ser posible también algún subdominio. Por ejemplo es.juntadeandalucia.medioAmbiente.interoperabilidad, donde es.juntadeandalucia es el nombre del dominio y medioAmbiente e interoperabilidad son nombres de subdominios.
- **artifactId**: identificará el nombre del proyecto sin incluir la versión. Por ejemplo ServicioDePrueba_CP (Por defecto el editor asignará el mismo valor que el nombre del proyecto).
- **version**: identificará la versión del proyecto, donde siempre se deberá incluir una nomenclatura de tres dígitos [X.Y.Z], siguiendo las pautas de versionado indicadas en este documento para el tipo de servicio en cuestión.

Es necesario cumplir la correcta definición de propiedades de Maven para poder realizar futuros versionados de los servicios.

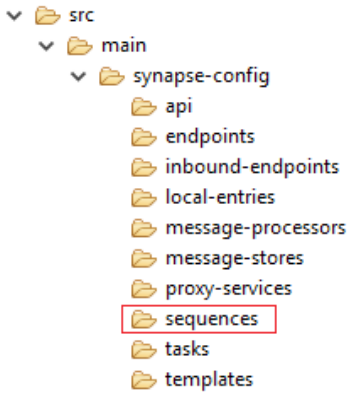
2.1.7.3. Recursos y artefactos

A continuación se muestran una serie de pautas genéricas que hay que tener en cuenta a la hora de desarrollar un nuevo servicio:

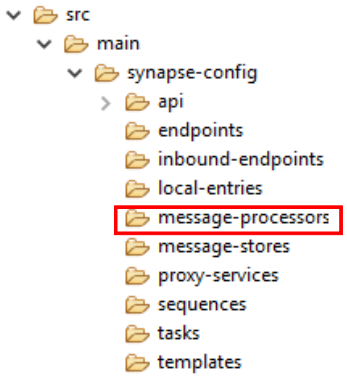
- Todos los recursos que no sean gestionados desde el registro deben encontrarse en su directorio correspondiente. Dentro de esos recursos se encuentran las secuencias, que deberán estar dentro de la carpeta *sequences,* los procesadores de mensajes, que deberán estar dentro de la carpeta *message-processors* y los almacenes de mensajes, que deberán estar dentro de la carpeta *message-stores.* El patrón que deben seguir todos los nombres de recursos es el siguiente:

  [SIGLAS_DEL_GRUPO_DEL_SERVICIO]_[SIGLAS_DEL_NOMBRE_DEL_SERVICIO]_[NOMBRE_DEL_RECURSO]_V[X.Y.Z]

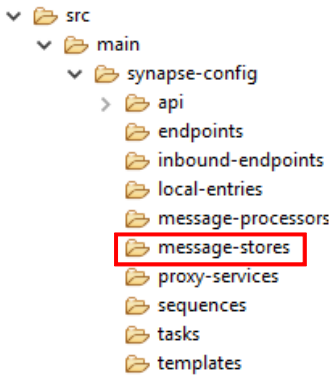
  - **[SIGLAS_DEL_GRUPO_DEL_SERVICIO]** serán las siglas del nombre del grupo en mayúscula.
  - **[SIGLAS_DEL_NOMBRE_DEL_SERVICIO]** serán la siglas del nombre del servicio en mayúsculas. Por ejemplo para el servicio ServicioDePrueba sería SDP.
  - **[NOMBRE_DEL_RECURSO]** será el nombre del recurso en formato CamelCase con la primera letra en mayúscula, y deberá plasmar a través del nombre cual es la función principal que se realiza al utilizar dicho recurso. A continuación se muestran los recursos más empleados en los desarrollos así como su ubicación. Estos recursos son:
    - Secuencias: se ubicaría en el siguiente directorio:



    - Message-Processor: se ubicaría en el siguiente directorio:



    - Message-Store: se ubicaría en el siguiente directorio:



- **[X.Y.Z]** hace referencia a los dígitos de la versión. Debe seguir las pautas indicadas en este documento para el tipo de servicio correspondiente.

- Los recursos que deban ser gestionados desde el registro, deberán almacenarse en la ruta lógica **gov:servicios/[TIPO_DE_SERVICIO]/[SIGLAS_DEL_GRUPO_DEL_SERVICIO]/[NOMBRE_DEL_SERVICIO]/v[X.Y.Z]/[TIPO_DEL_RECURSO]/[NOMBRE_DEL_RECURSO]**
  - **[TIPO_DEL_SERVICIO]** será el tipo de servicio en mayúsculas. Los valores más comunes serán **SOAP** y **REST**.
  - **[SIGLAS_DEL_GRUPO_DEL_SERVICIO]** serán las siglas del nombre del grupo en mayúscula.
  - **[NOMBRE_DEL_SERVICIO]** será el nombre del servicio según las normas de nomenclatura indicadas en el apartado 2.1.7.1 de este documento.
  - **[X.Y.Z]** hace referencia a los dígitos de la versión. Debe seguir las pautas indicadas en este documento para el tipo de servicio correspondiente.
  - **[TIPO_DEL_RECURSO]** será el nombre del tipo del recurso en plural que se desea almacenar en el registro. Por ejemplo endpoints, wsdls, parametros, javascripts, xslts, etc.
  - **[NOMBRE_DEL_RECURSO]** será el nombre del recurso según el estilo de escritura CamelCase con el primer carácter en mayúscula y podrá separarse por un guion bajo cuando se mencione también el nombre del servicio. Por ejemplo TransformacionDatosEspecificos_ServicioDePrueba.xsl
- Todas las propiedades (*<property>*) que se definan en el servicio deben dejar claro cual es el contenido que se almacenará en ellas.

2.1.7.4. Control de versiones del código

Se debe utilizar el gestor de control de versiones (Git o SVN) implantado en la Consejería para el control de las de versiones de los desarrollos. Las pautas generales de uso de esta herramienta podrán encontrarse en el documento ***GEN-LibroBlanco_obligatorio*** perteneciente a la Normativa de la Consejería.

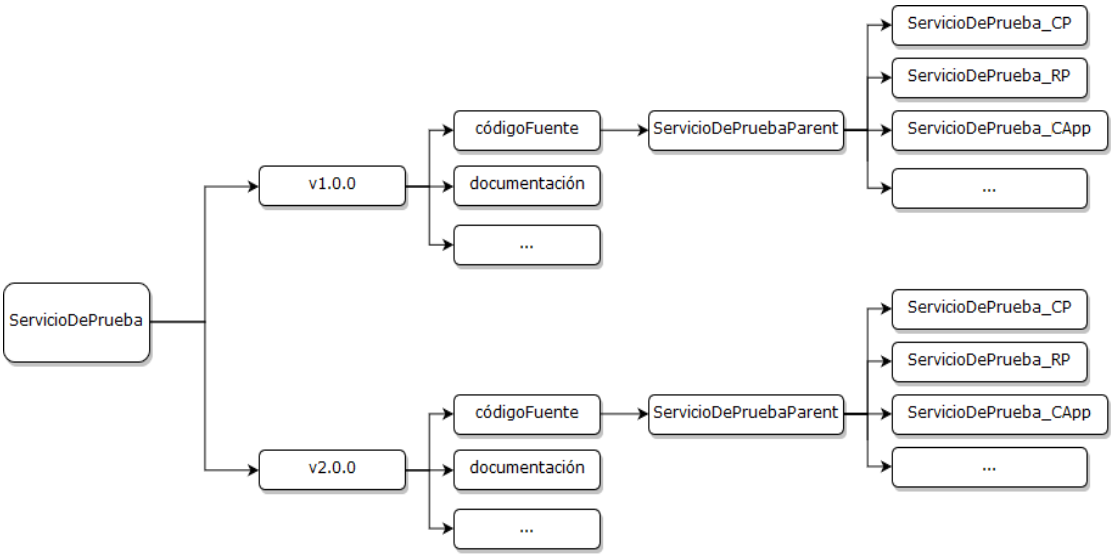
Siempre se debe tener actualizado el código del repositorio con la última versión del software. Existen ciertas particularidades que aplican a los desarrollos en WSO2 a la hora de gestionar las versiones del código:

- No es posible versionar solamente un artefacto actualizando su versión en el *artifact.xml,* ya que si se intentara desplegar dos servicios que contienen un artefacto con el mismo nombre, el componente WSO2 correspondiente no lo permitiría, ya que entiende que hay otro artefacto con el mismo nombre. Por ello, la mejor solución para resolver este escenario, es añadir la referencia de la versión también en el nombre del artefacto, es decir, se debería de seguir el siguiente patrón:

  [NOMBRE_DEL_ARTEFACTO]_V[X.Y.Z]

- Para desarrollar una lógica con cohesión alta y acoplamiento bajo, se debe aislar todos los artefactos y recursos de una versión a otra, es decir, es necesario crear un Maven Multi Module Project con cada versión del servicio.

  Por ejemplo, en un escenario en el que existe un servicio llamado *ServicioDePrueba* y cuenta con dos versiones del servicio, la estructura de carpetas en el repositorio debería quedar de la siguiente forma:



Es necesario seguir ciertos patrones para describir claramente cuales han sido las tareas acometidas a la hora de subir una nueva versión del software al repositorio:

- El **+** determinará que lo descrito es una adición.
- El ***** determinará que lo descrito es una modificación.
- El **–** determinará que lo descrito es una eliminación.

2.1.7.5. Log

Se deben tener en cuenta las siguientes consideraciones a la hora de establecer el registro de logs de la Plataforma de Interoperabilidad:

- Mostrar siempre información sobre posibles errores que hayan sido encontrados en el servicio.
- Mediación de mensajes: es obligatorio que todos los logs incluyan la propiedad MessageId (devolverá el identificador único del mensaje), con el fin de poder seguir la traza de la petición dentro del bus:

```
<property expression="get-property('MessageID')" name="CORRELATION_ID"
scope="default" type="STRING" xmlns:ns="http://org.apache.synapse/xsd"/>
```

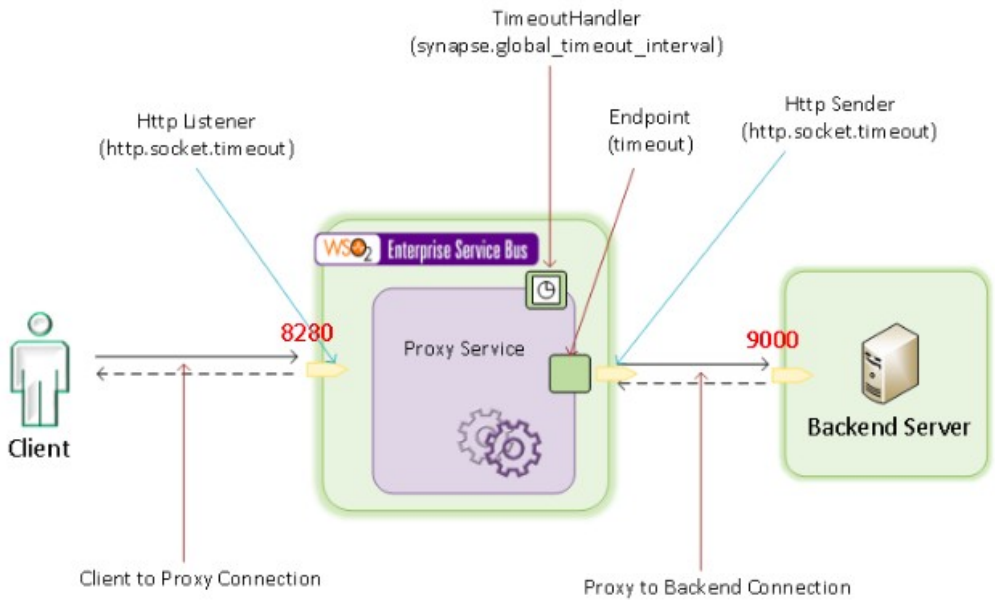
- En entornos de Producción, solo se debe utilizar el mediador Log con categoría **ERROR** en la secuencia de error (faultSequence). Los logs de la secuencia de entrada (inSequence) y la secuencia de salida (outSequence), deberán tener la categoría **DEBUG**.
- Incluir el mediador Log en la faultSequence para capturar información sobre los errores. Es posible utilizar las propiedades **ERROR_CODE**, **ERROR_MESSAGE**, **ERROR_DETAIL** y **ERROR_EXCEPTION** para obtener información del error. También se debe crear una propiedad para conocer el punto del flujo del servicio en el que se encuentra la

ejecución en cada momento y capturar esta propiedad cuando exista algún error, de forma que se facilite el conocer en que momento se produjo el error.

- En el entorno de desarrollo, hay que establecer el nivel global de log de la propiedad *org.apache.synapse.mediators.builtin.LogMediator* a **DEBUG**. Sin embargo, en el entorno de Producción, hay que establecer el nivel global de log de la propiedad *org.apache.synapse.mediators.builtin.LogMediator* a **INFO**.
- Se deben aislar los logs por servicio, para poder mediar solo los logs de un servicio determinado. Para poder lograr esto, se puede seguir la guía Enabling Logs for Services.

2.1.7.6. Timeout

En la siguiente imagen se ilustra el flujo típico del mensaje cuando un servicio de la Plataforma de Interoperabilidad está involucrado en una comunicación cliente-servidor. Existen dos conexiones diferentes que no dependen cada una de la otra: Cliente – Servicio de la PI y Servicio de la PI – Backend, por lo tanto, incluso si una conexión expira, la otra no se ve afectada.



En un entorno de Producción, se deben configurar los siguientes parámetros de timeout:

- **http.socket.timeout**: Es el timeout del socket del transporte de envío y escucha del Passthrough HTTP/HTTPS
  - Fichero de configuración: {EI_HOME}/conf/passthru-http.properties
  - Valor por defecto: 180000 (en milisegundos)
  - Valor a configurar: 180000 (en milisegundos)
- **Timeout del endpoint**: Es el timeout que debemos configurar a nivel del endpoint cuando definimos el servicio a publicar en la Plataforma de Interoperabilidad. Siempre se debe de seguir la fórmula:

  Timeout del socket > max(Timeout global endpoint, Timeout del endpoint individual)

  - Fichero de configuración: Propia configuración del endpoint de cada servicio
  - Valor por defecto: synapse.global_timeout_interval
  - Valor a configurar: Depende del caso de uso, típicamente 120000 (en milisegundos)

- **synapse.global_timeout_interval**: Es el timeout global de los endpoints. Synapse es la máquina de mediación que se encuentra por debajo del componente WSO2 EI, y se trata completamente de una máquina de mensajería asíncrona que no bloquea sus hilos de trabajo en la red I/O. En vez de ello, realiza los siguientes pasos:
  1. Registra una llamada de respuesta (call-back) para una petición específica y libera los hilos sin esperar la respuesta.
  2. Cuando una respuesta está disponible, la llamada de respuesta registrada es usada para relacionar ésta con la solicitud correspondiente para que se pueda realizar el procesamiento posterior. Si el backend no llegara a responder, se libera la llamada de respuesta registrada, como máximo después de la duración especificada en el parámetro **synapse.timeout_handler_interval** para prevenir posibles fugas de memoria.

  Es decir, el parámetro synapse.global_timeout_interval representa la duración que un mensaje de respuesta debe ser mantenida en el almacenamiento de llamadas de respuesta.

  - Fichero de configuración: {EI_HOME}/conf/synapse.properties
  - Valor por defecto: 120000 (en milisegundos). No es recomendable incrementar este valor
  - Valor a configurar: 120000 (en milisegundos)
- **synapse.timeout_handler_interval**: Es la duración entre la ejecución de dos ***TimeoutHandler***, tarea periódica que se lanzará para comprobar si las llamadas de respuestas registradas en el almacén de Synapse deben ser liberadas por haber superado el timeout del endpoint. Es decir, como máximo una llamada de respuesta registrada se almacenará en el almacén durante el timeout del endpoint + timeout del handler.

2.1.7.7. Política de reintentos

Se deben aplicar políticas de reintentos para confirmar cual es el estado del backend. Una llamada errónea al backend no debería provocar que la Plataforma de Interoperabilidad marcase ese backend en estado SUSPENDIDO (SUSPENDED), ya que en ese caso, durante 30 segundos las llamadas posteriores al backend no se realizarían porque el bus entiende que el backend no estaría disponible.

Para evitar que tras una sola llamada de error, el backend quede en estado SUSPENDED, debemos aplicar las siguientes configuraciones en la definición del endpoint del backend en el bus:

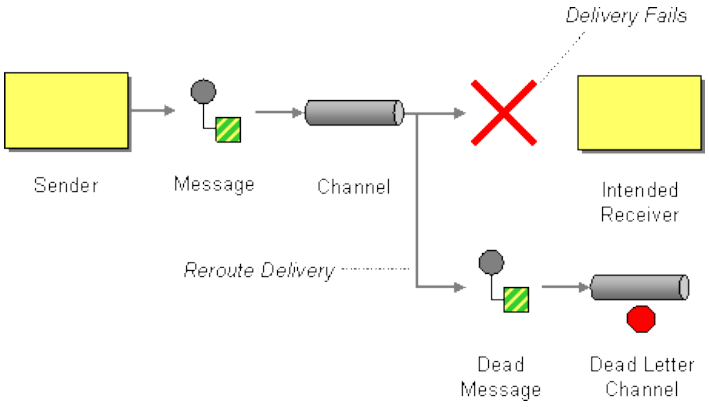
```
<timeout>
      <duration>.....</duration>
       <responseAction>.....</responseAction>
</timeout>

<markForSuspension>
      <errorCodes>.....,.....,.....</errorCodes>
      <retriesBeforeSuspension>.....</retriesBeforeSuspension>
      <retryDelay>.....</retryDelay>
</markForSuspension>

<suspendOnFailure>
      <errorCodes>.....,.....,.....</errorCodes>
      <initialDuration>.....</initialDuration>
      <progressionFactor>.....</progressionFactor>
      <maximumDuration>.....</maximumDuration>
</suspendOnFailure>
```

- En las configuraciones de <timeout> se indicará cual es el máximo tiempo que el bus esperará para obtener la respuesta del backend.
- En las configuraciones de <markForSuspension> se indicarán cuales son los códigos de error que si se dan, provocaran que el endpoint se marque en estado **TIMEOUT**.

- En las configuraciones de <suspendOnFailure> se indicarán cuales son los códigos de error que si se dan, provocaran que el endpoint se marque en estado **SUSPENDED**.

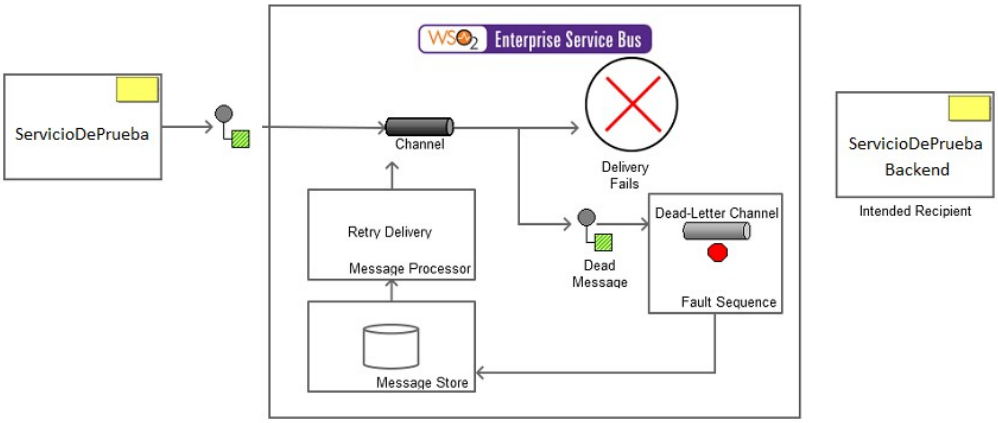
2.1.7.8. Aseguramiento del mensaje

Se debe de aplicar el patrón **Dead Letter Channel** para el aseguramiento en el envío del mensaje al backend, ya que si se produjera algún error en el envío o se sobrepasara el timeout, el mensaje se perdería y se le devolvería un mensaje de error al cliente.



En entornos de Producción, es necesario realizar más de un intento a la hora de entregar el mensaje al backend, por si el backend se encontrara caído o no pudiera atender la petición en ese momento. Para lograr este escenario, se deberá implementar el siguiente diseño cuando un mensaje entre por la faultSequence, ya que indicaría que se produjo algún comportamiento inesperado en la Plataforma de Interoperabilidad.

Como se puede observar, se debe usar el mediador Store para almacenar el mensaje fallido en el almacenamiento que se designe



en el Message Store (no se recomienda usar el almacenamiento en memoria para entornos de Producción, en su caso se puede usar almacenamiento externo como JMS). Posteriormente se irán procesando los mensajes almacenados para su reenvío al endpoint que se especifique con el Message Proccesor. En este procesamiento se puede indicar el intervalo de reintento de entrega del mensaje y el número de reintentos.

Se pueden obtener más detalles sobre la implementación del patrón Dead Letter Channel en el siguiente enlace.

2.1.7.9. Uso de librerías externas

Existe la posibilidad de extender las características disponibles del producto añadiendo nuevas librerías externas. En función del tipo de librería, se deberán guardar en un directorio diferente:

- **Librerías de componentes OSGI**: {EI_HOME}/repository/components/dropins
- **JAR que serán convertidos a componentes OSGI durante el arranque del servidor**: {EI_HOME}/repository/components/lib
- **JAR que serán convertidos a componentes de extensión del marco OSGI durante el arranque del servidor**: {EI_HOME}/repository/components/extensions

2.1.7.10. Codificación de caracteres

Todos los elementos de los Servicios Web de la Consejería, independientemente del sistema en el que se encuentren publicados, debe hacer uso del encoding UTF-8. Esto también incluye a los mensajes y parámetros entrantes y salientes de los servicios SOAP y REST.

2.2. Normas para servicios tipo SOAP

2.2.1. Archivo WSDL

Cada Servicio Web ha de disponer y publicar el correspondiente archivo WSDL (Web Service Description Language) que ha de ser auto documentado, es decir, ha de contener detalle explicativo del servicio, de las operaciones y todos los tipos de datos que describa.

En el caso de que el Servicio Web incluya elementos como seguridad, addressing, etc., el WSDL incluirá las políticas empleadas según se define en la especificación *WS-Policy*.

Los **namespaces** dentro de los WSDLs y esquemas empleados dentro de los Servicios Web, se normalizan dentro de MADEJA (Marco de Desarrollo de la Junta de Andalucía). Los namespaces deberán incluir una referencia a la Junta de Andalucía, Consejería, aplicación, servicio, componente y número de versión.

Para la definición de los namespaces se puede optar por emplear URIs basadas en URLs o URN (Unified Resources Name). El formato recomendado por MADEJA y el obligatorio en los desarrollos en Consejería es el de URN, al ser más claro y requerir menos procesos de normalización que el de URLs.

En concreto, el formato y la descripción de los campos se recogen en la tabla siguiente:

Formato		
urn:juntadeandalucia:medioAmbiente:(aplicación):(versión):(servicio):(componente -opc)		
Descripción de los campos.		
Elemento	**Obligatorio**	**Descripción**
juntadeandalucia	Sí	Referencia a la Junta de Andalucía
medioAmbiente	Sí	Referencia a la Consejería
Aplicación	Sí	Nombre de la aplicación a la que pertenece el servicio.
Versión	Sí	Número de versión
Servicio	No	Nombre del servicio.
Componente	No	Nombre del elemento o componente

Tabla 7: Formato y descripción de los campos del URN en servicios web SOAP de la Consejería

2.2.2. Definición de servicios

Los Servicios Web desarrollados han de seguir la aproximación *Contract-first,* esto es, primero se generará el WSDL y después se codificará el servicio. Por tanto, de forma previa a implementar el código, habrá que definir las operaciones, métodos y datos del negocio del servicio como fase inicial del análisis.

2.2.3. Patrón de diseño XML Schema

Los esquemas XML que se utilicen junto a los descriptores de contratos de servicios web (WSDL) tendrán que cumplir 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.

A la hora de diseñar el schema XSD, se han de crear tipos y elementos globales (a nivel raíz) para poder reutilizarlos, tanto a nivel de elementos XML como del WSDL. Un elemento global es cualquier hijo de un nodo o una referencia directa a uno de ellos.

2.2.4. Definición de atributos y operaciones

No se permiten servicios que expongan operaciones con parámetros de entrada y/o salida de tipo cadena de texto que lleven incrustado un archivo XML con toda la lógica asociada.

Se ha de 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.

2.2.5. Intercambio de documentos a través de los servicios

La trasmisión de ficheros mediante Servicios Web se debe realizar de una forma optimizada, haciendo uso de MTOM, o bien como *attachments*.

Siempre que sea posible, debe evitarse la transmisión de ficheros en el *body* del mensaje.

2.2.6. Versionado de servicios y de esquemas de datos

2.2.6.1. Definiciones y renombrado de versiones

A continuación se detallan los diversos tipos de cambios que se han establecido para un Servicio Web (ejemplos de versionado pueden encontrarse en el **Anexo I**):

- **Revisión**: Cambio que se produce cuando se arregla algún bug o problema interno. Estos cambios no alteran el contrato del servicio.
- **Versión Menor**: Son cambios que mantienen la compatibilidad, es decir, son cambios en la implementación que no modifican la funcionalidad actual en el servicio y que por tanto no obligan a los consumidores de dicho servicio a volver implementar la interfaz.
- **Versión Mayor**: Son cambios que hacen incompatibles el servicio, es decir, son cambios que modifican la funcionalidad del servicio y que, por tanto, obligan a volver a implementar la interfaz a los consumidores de dicho servicio.
- Es importante destacar que un cambio de versión de un Esquema de Datos utilizado por un servicio implicará el cambio de versión también del servicio.

Como se ha comentado en el apartado de *Namespaces*, el nombre del servicio incluye el número de versión, que se incrementará en caso de que los cambios den lugar a una *versión mayor* y no variará ante los cambios con compatibilidad hacia atrás (cambios menores o revisión). Ejemplo**:** v01/nombreServicio → v02/nombreServicio

2.2.6.2. Adaptación de los clientes

Para permitir que los clientes de los servicios se adapten ante cambios de versión se seguirán las siguientes reglas:

- Los cambios de versiones menores no implicarán cambio de versión del servicio pero han de ser documentados en el WSDL y requerirán una actualización del contrato del servicio. El proveedor del servicio tendrá que comunicarlos a los clientes del servicio para que estos puedan decidir si hacer uso de las funcionalidades añadidas.
- Los cambios de versión mayor implicarán cambio de versión del servicio. Para permitir la adaptación de los clientes a las nuevas versiones se permitirá la existencia de hasta dos versiones del servicio funcionando en paralelo. Se garantizará el acceso al servicio anterior durante un plazo máximo de dos meses, tiempo en el que el cliente del servicio deberá adaptarse para consumir la nueva versión del servicio.
- Los cambios de versiones de los servicios han de ser notificados a los clientes, tras la correspondiente actualización del WSDL y del Contrato del Servicio. En casos excepcionales en los que el cliente no pueda adaptarse, podrán mantenerse dos versiones en paralelo del servicio.

2.2.7. Versionado de especificación

Será necesario desarrollar utilizando siempre la misma versión de especificación o versiones compatibles, puesto que no todas las especificaciones son compatibles hacía atrás. Por ejemplo, SOAP 1.2 no es totalmente compatible con SOAP 1.1 o SOAP 1.0. Un nodo que cumpla con SOAP 1.1 generara un mensaje SOAP Fault indicando que la versión no coincide si le llega un mensaje SOAP 1.2. Un nodo utilizando SOAP 1.2 tendrá la opción de procesar el mensaje, o generar un SOAP Fault.

2.2.8. Equilibrio entre servicios y operaciones

Es habitual encontrarse con las siguientes situaciones a la hora de desarrollar servicios web:

- Un único servicio, con multitud de operaciones.
- Servicios por operación.

Ambos extremos son perjudiciales. Se deben diseñar servicios web con las responsabilidades bien repartidas, que sean cohesivos, extensibles, escalables y reutilizables.

Puede tomarse como un buen punto de partida el definir un Servicio Web por cada servicio de negocio identificado.

2.2.9. Content-Type

El campo Content-type de la cabecera del mensaje, define el tipo de datos de cada parte como type/subtype.

Uno de los valores más comunes que puede adoptar el campo content-type es "text/xml". Sin embargo, no está soportado en SOAP 1.2, por lo que el valor que debe configurarse, habitualmente, para el desarrollo de Servicios Web es "application/soap" o "application/soap+xml". Para el uso de algunos estándares que se apoyan en SOAP, es posible que deban configurarse otros valores, como es el caso de MTOM.

2.2.10. Gestión de errores en servicios web SOAP

La respuesta de un servicio SOAP ante un error debe implementarse de dos formas diferentes según el error que se produzca:

- **Errores desvinculados de la funcionalidad del servicio**: esta implementación incluiría los errores de comunicación, los errores derivados de la validación del mensaje y los errores internos de la aplicación. Todos ellos serán implementados a través de soapFault, en su versión 1.2. (http://schemas.xmlsoap.org/soap/envelope/).
- **Errores funcionales**: todos aquellos derivados del tratamiento de la información obtenida del mensaje y que derivan en problemas como inconsistencia de datos, datos maestros no registrados en el proveedor, etc. Estos errores se implementarán a través de un formato definido para ello en cada uno de los contratos de integración.

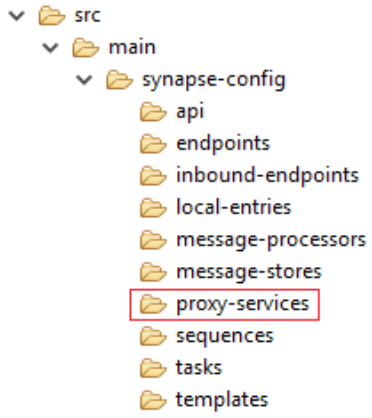
2.2.11. Desarrollo de servicios SOAP

En este apartado se recogen las normas específicas que deben cumplirse a la hora de desarrollar un servicio SOAP para ser publicado en la Plataforma de Interoperabilidad.

2.2.11.1. Recursos y artefactos

A continuación se muestran una serie de pautas que hay que tener en cuenta a la hora de desarrollar un nuevo servicio:

- El servicio deberá estar dentro de la carpeta *proxy-services* y seguir el patrón **[TIPO_DE_VISIBILIDAD]_[TIPO_DE_SEGURIDAD]_[GRUPO_DEL_SERVICIO]_[NOMBRE_DEL_SERVICIO]_V[X.Y.Z]**
  - **[TIPO_DE_VISIBILIDAD]** identifica si el servicio es interno (**INT**), es decir, solo puede ser consumido dentro de la Red Corporativa de la Junta de Andalucía (RCJA), o externo (**EXT**), que indicaría que puede ser consumido fuera de la RCJA.
  - **[TIPO_DE_SEGURIDAD]** identifica si el servicio posee algún tipo de seguridad de acceso (UserrnameToken, MutualSSL, etc). El servicio puede ser público (**PUB**), indicando que no existe ningún mecanismo de seguridad de consumo del servicio, o privado (**PRV**), indicando que si se posee un mecanismo de seguridad de consumo del servicio.
  - **[SIGLAS_DEL_GRUPO_DEL_SERVICIO]** serán las siglas del nombre del grupo en mayúsculas, y se indicará cuando exista un grupo bien diferenciado que pueda englobar a varios servicios de índole similar. Por ejemplo, una posible agrupación se daría cuando una consejería, organismo o sistema proporcionaran más de un servicio.
  - **[NOMBRE_DEL_SERVICIO]** seguirá las pautas de nomenclatura mencionadas en el apartado 2.1.7.1 de este documento.
  - **[X.Y.Z]** hace referencia a los dígitos de la versión y seguirá las normas de versionado correspondientes al versionado de servicios SOAP mencionadas en este documento.


src
main
synapse-config
api
endpoints
inbound-endpoints
local-entries
message-processors
message-stores
proxy-services
sequences
tasks
templates

2.3. Normas para servicios tipo REST

2.3.1. Uso de los métodos HTTP

A continuación, se realizan indicaciones acerca del uso que ha de realizarse de los métodos principales de este protocolo en los Servicios Web de la Consejería:

- El método **GET** sólo debe usarse para obtener información del servidor, de acuerdo a los estándares de HTTP. Este método no debe cambiar el estado del servidor, es decir, no debe hacer ninguna modificación a ningún recurso del servidor. Debe usarse este método, por tanto, para realizar consultas sobre recursos. Dado que su ejecución no modifica estado en el servidor, es recomendable su uso para el cacheo de datos.

  Ejemplos:

  - Obtener listado de elementos → GET /api/v1.0/elementos/
  - Obtener elemento concreto → GET /api/v1.0/elementos/{idElemento}

- El uso del método **POST** debe limitarse al alta de nuevas entidades en el sistema. Debe enviarse el recurso a añadir en el cuerpo del mensaje (Body Param)

  Ejemplo:

  - Crear un nuevo elemento → POST /api/v1.0/elementos/
    Content-Type: application/json
    Accept: application/json
    {
    <datos en formato JSON>
    }

- El método **PUT** debe usarse para modificar recursos ya existentes. Esta modificación se realizará en modo foto, es decir, si alguno de los atributos del recurso a modificar no viene indicado en el mensaje de entrada, tendrá que interpretarse como que debe ser borrado, no mantenerlo con el valor actual. Debe enviarse el recurso a modificar en el cuerpo del mensaje (Body Param).
  Ejemplo:
  - Modificar un elemento → PUT /api/v1.0/elementos/(idElemento)
    Content-Type: application/json
    Accept: application/json
    {
    <datos actualizados en formato JSON>
    }

- El método **DELETE** es el único que debe utilizarse para borrar un recurso del servidor, no estando permitido emplear otros métodos para este fin, no se deben utilizar GET o POST por ejemplo.
  Ejemplo:

  - Eliminar un elemento → DELETE /api/v1.0/elementos/(idElemento)

2.3.2. Nomenclatura de los URIs

La URI será el identificador único de cada recurso del sistema REST. Han de seguir una serie de reglas:

- Han de estar orientadas a la exposición de recursos y no de operaciones o eventos.

  ../api/v1/usuarios/{dni}/informes/tipoInforme/{idInforme}

- Nomenclatura de campos y parámetros CamelCase con primer carácter en minúscula (lowerCamelCase)

- Los nombres de URI no deben implicar una acción, por lo tanto no se podrán usar verbos en ellos.
- Han ser únicas, es decir, no puede haber más de una URI para identificar un mismo recurso.
- Para acceder a la información de un recurso concreto, deberá hacerse uso de identificadores insertados como parámetros dentro del path (Path Params). Estos identificadores deben ser únicos dentro del ámbito del recurso al que se quiere acceder.

  ../api/v1/usuarios/00000000T

- Han de incluir la versión de la API. Esta versión será sólo modificada en caso de cambios mayores. Para más información, consultar el apartado 2.3.3 de este documento.

  ../api/v1/elementos/

- Han de ser independientes del formato.
  - Por ejemplo, la URI */facturas/25.pdf* no sería una URI correcta, ya que se está indicando la extensión pdf en la misma.

    Para el recurso facturas con el identificador 25, la siguiente URI sería la correcta, independientemente de que se vaya a consultarla en formato pdf, epub, txt, xml o json: */facturas/25*
- Han de mantener una jerarquía lógica según el aspecto de negocio que trate.
  - Por ejemplo, la URI */facturas/25/clientes/007* no sería una URI correcta, ya que no sigue una jerarquía lógica de negocio ya que la factura está asociada al cliente y no al revés.
  - Para el recurso factura con el identificador 25 del cliente 007, la siguiente URI sería la correcta: */clientes/007/facturas/25*.
- Los filtrados de información de un recurso no se hacen en la URI. Se realiza concatenando atributos.
  - Para filtrar, ordenar, paginar o buscar información en un recurso, se ha hacer una consulta sobre la URI, utilizando parámetros HTTP (Query Params) en lugar de Path Params.

    Por ejemplo, la URI */facturas/orden/desc/fecha-desde/2007/pagina/2* sería incorrecta ya que el recurso de listado de facturas sería el mismo pero se utilizaría una URI distinta para filtrarlo, ordenarlo o paginarlo.

    La URI correcta en este caso sería */facturas?fecha-desde=2007&orden=DESC&pagina=2*
- Las fechas han tener el siguiente formato: YYYY[MM[DD[HH[mm[ss]]]]]
- Aquellas consultas que generan como respuesta un listado de recursos deben tener en cuenta las siguientes cuestiones:
  - Facilitar la posibilidad de filtrar y paginar la respuesta. Para la paginación se tendrá que añadir a la solicitud los atributos "page" y "pageSize" para indicar la pagina y el tamaño de la misma respectivamente.
  - La respuesta a estas consultas añadirán en la cabecera los parámetros "X-Total-Count" (número total de resultados encontrados), "X-Page-Size" (tamaño de paginas), "X-Page" (página actual), "Link" (enlaces a página primera, última, siguiente y anterior).
  - Los listados que permitan establecer una ordenación han de indicar cuales son los campos por los que se puede ordenar. El orden natural será de menor a mayor y se notificará mediante el parámetro "order" el campo de información de este listado por el cual se pretende ordenar. Si fuese necesario establecer un orden en base a varios campos de información se concatenarán estos mediante "," reservando la posibilidad de añadir el carácter "-" delante del campo para establecer un orden inverso al natural.
  - Del mismo modo, mediante el parámetro "fields" se podrá indicar los campos de información mínimos requeridos en la respuesta. Este parámetro es necesario ya que se establece como norma que, a la hora de acceder recursos, se devuelva la información funcional mínima requerida en cada consulta, de manera que si esta fuera necesaria ampliarla sea posible solicitar aquellos que no se obtengan por defecto. Para concatenar campos de información a

añadir, se hará uso del carácter ",". Se puede indicar el valor "all" a este parámetro de modo que la respuesta debe venir con todos los campos posibles indicados.

2.3.3. Versionado

El número de la versión del servicio constará de tres dígitos que seguirán las siguientes reglas para su modificación:

- **Revisión**: Cambio que se produce cuando se arregla algún bug o problema interno. Estos cambios no alteran el contrato del servicio.
- **Versión Menor**: Son cambios que mantienen la compatibilidad, es decir, son cambios en la implementación o en la definición que no modifican la funcionalidad actual en el servicio y que por tanto no obligan a los consumidores de dicho servicio a volver implementar la interfaz.
- **Versión Mayor**: Son cambios que hacen incompatibles el servicio, es decir, son cambios que modifican la funcionalidad del servicio o la definición de forma que no sea compatible con la anterior y que, por tanto, obligan a volver a implementar la interfaz a los consumidores de dicho servicio.

La versión del servicio se indicará en los siguientes puntos:

- **URI del servicio**: sólo se indicará el valor de la Versión Mayor. Seguirá el formato "v[X]". Se utilizará para mantener vivas dos versiones incompatibles, pudiendo ser posible el acceso a ambas de forma simultánea.
  - Ejemplo: ../api/v1/usuarios/{idUsuario}/documentos/tipoDocumento/{idDocumento}
- **Cabecera "version"**: cada vez que se acceda a un servicio REST, se tendrá que enviar una cabecera HTTP con nombre "version" donde se indique la versión del servicio a la que se quiere acceder. El formato del contenido de la cabecera será **[X.Y.Z]**. El receptor del mensaje realizará el procesamiento correspondiente a la versión solicitada.
- **Nombres de los componentes de la plataforma**: deben seguirse las normas del apartado 2.1.7.4 de este documento para el versionado de elementos desplegados en la Plataforma de Interoperabilidad.

Como se ha mencionado, no todos los cambios en un Servicio REST tienen impacto sobre los consumidores del mismo, a estos cambios se les suele denominar **cambios retrocompatibles**. En caso de que el Servicio REST sufra cambios de este tipo, aunque en los componentes desplegados en la Plataforma de Interoperabilidad sí se reflejará este cambio de versión, la URI del servicio se mantendrá sin alterar, de forma que no deban realizarse cambios en los sistemas que invocan al servicio. Será obligatorio informar a los consumidores de los Servicios REST de los nuevos cambios para que los tengan en cuenta, por si quisieran hacer uso de ellos.

Este es un listado de cambios en un Servicio REST que NO deberían afectar a los consumidores:

- Añadir nuevas operaciones al servicio no existentes hasta el momento (PUT, POST...).
- Añadir parámetros de entrada (Query Params o Body Params) opcionales a operaciones sobre recursos ya existentes.
- Modificar parámetros de entrada de obligatorios a opcionales.
- Añadir nuevas propiedades en la información de un recurso que devuelve el servidor. Ejemplo: ahora a un recurso Persona que anteriormente estuviese compuesto por DNI y nombre, se le añade un nuevo campo edad.

Por otro lado existen cambios que sí afectan a los consumidores de los servicios y que obligan a **modificar la versión de la URI del servicio REST**. Para permitir la adaptación de los clientes de esta nueva versión del servicio se tendrá que mantener en paralelo la versión anterior del servicio durante un periodo de tiempo establecido. Durante este tiempo se adaptarán los consumidores del servicio a la nueva versión. Se permitirá la existencia de hasta dos versiones del servicio funcionando en paralelo.

Se indican un conjunto de cambios que sí obligan a versionar la API:

- Eliminar operaciones sobre un recurso (POST, PUT,...)
- Añadir nuevos parámetros de entrada (Query Params y Body Params) obligatorios.
- Modificar parámetros de entrada (Query Params y Body Params) de opcional a obligatorio.

- Modificar un parámetro en operaciones ya existentes. También aplicable a la eliminación de parámetros. Ejemplos: al consultar un recurso ya no se devuelve determinado campo; un campo que antes era una cadena de caracteres, ahora es numérico.
- Añadir nuevas respuestas en operaciones ya existentes. Ejemplo: añadir nuevos códigos de error HTTP.

2.3.4. JSON Schema

Todos los recursos a intercambiar en la mensajería de los servicios REST definidos en la Consejería deberán de hacer uso del formato de texto JSON.

Para garantizar que estos mensajes cumplen con la definición del servicio, debe generarse un fichero JSON Schema por cada uno de ellos, de forma que pueda ser utilizado para la validación de la estructura del mensaje.

2.3.5. Content-Type

El campo Content-type de la cabecera del mensaje, define el tipo de datos de cada parte como type/subtype.

En el caso de los servicios REST, esta cabecera deberá contener el valor "application/json".

2.3.6. Gestión de errores en servicios web REST

La respuesta de un servicio RestFull ante un error debe aportar información breve y concisa sobre el motivo del mismo al sistema origen. La estructura debe incluir siempre los siguientes elementos:

- **Estado**: Código del estado HTTP. Debe aparecer siempre, incluyendo respuestas sin error.
- **Tipo**: Identificador del origen del mensaje.
- **Código**: Código interno del error (definido por el sistema destino o la OTI)
- **Descripción**: Texto descriptivo del error

HTTP define un conjunto de códigos de status que pueden ser devueltos por la API. A continuación se indica una lista de los códigos a utilizar:

- **200 OK** - Respuesta a un exitoso GET, PUT o DELETE. Puede ser usado también para un POST que no resulta en una creación.
- **201 Created** – [Creada] Respuesta a un POST que resulta en una creación. Debería ser combinado con un encabezado Location, apuntando a la ubicación del nuevo recurso.
- **400 Bad Request** – [Petición Errónea] La petición está mal formada, como por ejemplo, si el contenido no fue bien parseado.
- **401 Unauthorized** – [Desautorizada] Cuando los detalles de autenticación son inválidos o no son otorgados. También útil para disparar un popup de autorización si la API es usada desde un navegador.
- **403 Forbidden** – [Prohibida] Cuando la autenticación es exitosa pero el usuario no tiene permiso al recurso en cuestión.
- **404 Not Found** – [No encontrada] Cuando un recurso no existente es solicitado.
- **405 Method Not Allowed** – [Método no permitido] Cuando un método HTTP que está siendo pedido no está permitido para el usuario autenticado o no se encuentra implementado para el servicio en cuestión.
- **429 Too Many Requests** – [Demasiadas peticiones] Cuando una petición es rechazada debido a la tasa límite.
- **500 Internal Server Error** – [Error interno] Cuando se produce un error interno en el servidor ajeno a la naturaleza del servicio.

- **503 Service Unavailable** – [Servicio no disponible] Cuando el servicio no puede responder a la petición por estar congestionado o realizando tareas de mantenimiento.
- **504 Gateway Timeout** – [Tiempo de espera superado] Cuando el servicio actúa de intermediario y no ha recibido a tiempo una respuesta por lo que no puede responder adecuadamente al sistema origen.

Todos los errores funcionales que puedan darse en un sistema destino deberán ser recogidos bajo el código 500. El listado de *code* que especifica cada uno de los posibles errores funcionales debe estar indicado en los correspondientes documentos técnicos .

2.3.7. Documentación de Servicios REST

La documentación de los Servicios REST se podrá realizar siguiendo la plantilla ***OTI_INT_Plantilla Contrato Servicio Web*** o mediante la herramienta propia desplegada en la Plataforma de Interoperabilidad para la publicación de definiciones de API REST (pendiente de determinar).

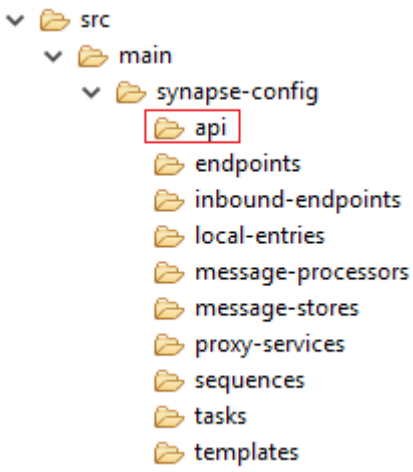
2.3.8. Desarrollo de servicios REST

En este apartado se recogen las normas específicas que deben cumplirse a la hora de desarrollar un servicio SOAP para ser publicado en la Plataforma de Interoperabilidad.

2.3.8.1. Recursos y artefactos

A continuación se muestran una serie de pautas que hay que tener en cuenta a la hora de desarrollar un nuevo servicio:

- El servicio deberá estar dentro de la carpeta *api* y seguir el patrón **[TIPO_DE_VISIBILIDAD]_[TIPO_DE_SEGURIDAD]_[GRUPO_DEL_SERVICIO]_[NOMBRE_DEL_SERVICIO]_V[X.Y.Z]**
  - **[TIPO_DE_VISIBILIDAD]** identifica si el servicio es interno (**INT**), es decir, solo puede ser consumido dentro de la Red Corporativa de la Junta de Andalucía (RCJA), o externo (**EXT**), que indicaría que puede ser consumido fuera de la RCJA.
  - **[TIPO_DE_SEGURIDAD]** identifica si el servicio posee algún tipo de seguridad de acceso (UserrnameToken, OAuth, etc). El servicio puede ser público (**PUB**), indicando que no existe ningún mecanismo de seguridad de consumo del servicio, o privado (**PRV**), indicando que si se posee un mecanismo de seguridad de consumo del servicio.
  - **[SIGLAS_DEL_GRUPO_DEL_SERVICIO]** serán las siglas del nombre del grupo en mayúsculas, y se indicará cuando exista un grupo bien diferenciado que pueda englobar a varios servicios de índole similar. Por ejemplo, una posible agrupación se daría cuando una consejería, organismo o sistema proporcionaran más de un servicio.
  - **[NOMBRE_DEL_SERVICIO]** seguirá las pautas de nomenclatura mencionadas en el apartado 2.1.7.1 de este documento.
  - **[X.Y.Z]** hace referencia a los dígitos de la versión y seguirá las normas de versionado correspondientes al versionado de servicios REST mencionadas en este documento.



3. ANEXO I. Ejemplos de versionados de servicios web

A continuación se recogen diferentes ejemplos de casuísticas de versionado de servicios web.

3.1. Versión menor

Las versiones menores contienen pequeñas modificaciones que afectan al contrato del Servicio pero mantienen la compatibilidad con la versión anterior. Ejemplos de estas modificaciones son:

Inclusión de un atributo no obligatorio a los datos de entrada

```
<xsd:complexType name="documento">
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="identifier"
                     nillable="false" type="xsd:string">
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="documento">
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="identifier"
                     nillable="false" type="xsd:string">
        </xsd:element>
        <xsd:element maxOccurs="1" minOccurs="0" name="description"
                     nillable="false" type="xsd:string">
    </xsd:sequence>
```

Figura 1: Versionado: Inclusión de un atributo no obligatorio a los datos de entrada (Versión menor)

Añadir un nuevo método al Servicio

```
<portType name="NombreServiov01">
    <operation name="operacion1">
        <input message="tns:operacion1"/>
        <output message="tns:returns_resultado1"/>
    </operation>
</portType>
```

```
<portType name="NombreServiov01">
    <operation name="operacion1">
        <input message="tns:operacion1"/>
        <output message="tns:returns_resultado1"/>
    </operation>
    <operation name="operacion2">
        <input message="tns:operacion2"/>
        <output message="tns:returns_resultado2"/>
    </operation>
</portType>
```

Figura 2: Versionado: Añadir un nuevo método al Servicio(Versión menor)

3.2. Versión mayor

Contienen modificaciones incompatibles con el contrato anterior. El cambio afecta a la compatibilidad hacia atrás del servicio. Ejemplos de estas modificaciones son:

Eliminación de un método.

```
<portType name="NombreServiov01">
    <operation name="operacion1">
        <input message="tns:operacion1"/>
        <output message="tns:returns_resultado1"/>
    </operation>
    <operation name="operacion2">
        <input message="tns:operacion2"/>
        <output message="tns:returns_resultado2"/>
    </operation>
</portType>
```

```
<portType name="NombreServiov02">
    <operation name="operacion1">
        <input message="tns:operacion1"/>
        <output message="tns:returns_resultado1"/>
    </operation>
</portType>
```

Figura 3: Versionado: Eliminación de un método (Versión mayor)

Renombrado de un método.

```
<portType name="NombreServiov01">
    <operation name="operacion1">
        <input message="tns:operacion1"/>
        <output message="tns:returns_resultado1"/>
    </operation>
</portType>
```

```
<portType name="NombreServiov02">
    <operation name="operacion1_renombrada">
        <input message="tns:operacion1"/>
        <output message="tns:returns_resultado1"/>
    </operation>
</portType>
```

Figura 4: Versionado: Renombrado de un método (Versión mayor)

Cambio en los parámetros de un método.

```
<portType name="NombreServiov01">
    <operation name="operacion1">
        <input message="tns:operacion1"/>
        <output message="tns:returns_resultado1"/>
    </operation>
</portType>
```

```
<portType name="NombreServiov02">
    <operation name="operacion1">
        <input message="tns:operacion1"/>
        <output message="tns:returns_resultadoX"/>
    </operation>
</portType>
```

Figura 5: Versionado: Cambio en los parámetros de un método (Versión mayor)

Cambios en un tipo de datos existente dentro del esquema.

```
<xsd:complexType name="documento">
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="identifier"
                     nillable="false" type="xsd:string">
        </xsd:element>
        <xsd:element maxOccurs="1" minOccurs="0" name="description"
                     nillable="false" type="xsd:string">
    </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="documento">
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="identifier"
                     nillable="false" type="xsd:int">
        </xsd:element>
        <xsd:element maxOccurs="1" minOccurs="0" name="description"
                     nillable="false" type="xsd:string">
    </xsd:sequence>
</xsd:complexType>
```

Figura 6: Versionado: Cambios en un tipo de datos existente dentro del esquema (Versión mayor)

Transformar un campo opcional a obligatorio.

```
<xsd:complexType name="documento">
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="identifier"
                     nillable="false" type="xsd:string">
        </xsd:element>
        <xsd:element maxOccurs="1" minOccurs="0" name="description"
                     nillable="false" type="xsd:string">
    </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="documento">
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="identifier"
                     nillable="false" type="xsd:string">
        </xsd:element>
        <xsd:element maxOccurs="1" minOccurs="1" name="description"
                     nillable="false" type="xsd:string">
    </xsd:sequence>
</xsd:complexType>
```

Figura 7: Versionado: Transformar un campo opcional a obligatorio (Versión mayor)

4. ANEXO II. Ejemplos de peticiones y respuestas SOAP

Petición sin seguridad

Se describe como sería la estructura de una petición a un servicio sin seguridad:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="urn:juntadeandalucia:medioAmbiente:comprobarconectividad:1.0">
        <soapenv:Header/>
        <soapenv:Body>
                <tns:comprobarconectividadRequest>
                        <url>https://prueba.com</url>
                </tns:comprobarconectividadRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 8: Petición de ejemplo de un servicio web SOAP sin seguridad

Petición con seguridad UsernameToken

Se describe como sería la estructura de una petición a un servicio con seguridad de UsernameToken con tipo PasswordDigest:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="urn:juntadeandalucia:medioAmbiente:comprobarconectividad:1.0">
        <soapenv:Header>
                <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
                        <wsu:Timestamp wsu:Id="TS-E08B253BCFD8F75F7315269225857502">
                                <wsu:Created>2018-06-18T06:41:37.372Z</wsu:Created>
                                <wsu:Expires>2018-06-18T06:42:37.372Z</wsu:Expires>
                        </wsu:Timestamp>
                        <wsse:UsernameToken wsu:Id="UsernameToken-E08B253BCFD8F75F7315269225833601">
                                <wsse:Username>prueba</wsse:Username>
                                <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordDigest">wPDd0oRfxFnjc8ck1kHQp16fA1s=</wsse:Password>
                                <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary">cW14J8f7CINYyY0wl2HYag==</wsse:Nonce>
                                <wsu:Created>2018-06-18T06:41:32.985Z</wsu:Created>
        </wsse:UsernameToken>
    </wsse:Security>
        </soapenv:Header>
        <soapenv:Body>
                <tns:comprobarconectividadRequest>
                        <url>https://prueba.com</url>
                </tns:comprobarconectividadRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 9: Petición de ejemplo de un servicio web SOAP con seguridad UsernameToken

Respuesta correcta

Se describe como sería la estructura de una respuesta correcta sin fallos:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="urn:juntadeandalucia:medioAmbiente:comprobarconectividad:1.0">
        <soapenv:Header/>
        <soapenv:Body>
                <tns:comprobarconectividadResponse>
                        <existeConexion>Si</existeConexion>
                        <codigoEstado>200</codigoEstado>
                </tns:comprobarconectividadResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 10: Respuesta de ejemplo de un servicio web SOAP sin errores

Respuesta con error SoapFault

Se describe como sería la estructura de una respuesta con errores SOAP (SoapFault):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode xmlns:tns="http://www.w3.org/2003/05/soap-envelope">tns:Receiver</faultcode>
         <faultstring>Error connecting to the back end</faultstring>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

Figura 11: Respuesta de ejemplo de un servicio web SOAP con error SoapFault

Respuesta con error funcional

Se indica un ejemplo de una posible respuesta con errores funcionales:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="urn:juntadeandalucia:medioAmbiente:comprobarconectividad:1.0">
        <soapenv:Header/>
        <soapenv:Body>
                <tns:comprobarconectividadResponse>
                        <functional-error>
                                <code>500</code>
                                <type>Conection Error</type>
                                <description>Error producido en la conexión con la URL</description>
                                <timestamp>2018-06-18T16:20:30.978Z</timestamp>
                        </functional-error>
                </tns:comprobarconectividadResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 12: Respuesta de ejemplo de un servicio web SOAP con error funcional