Junta de Andalucía

Consejería de la Presidencia, Interior, Diálogo Social y Simplificación Administrativa
Agencia Digital de Andalucía

Dirección General de Estrategia Digital

Financiado por la Unión Europea

Normas de desarrollo de servicios web

Normas y buenas prácticas para el desarrollo de servicios web

HOJA DE CONTROL DEL DOCUMENTO

Información del Documento			
Título	Normas de desarrollo de servicios web		
Asunto	Normas y buenas prácticas para el desarrollo de servicios web		
Nombre del fichero	OT-I_NORM_Normativas para el Desarrollo de Servicios Web_v01r04.odt		
Versión	<v01r04>	**Fecha versión**	12/09/2024
		N.º Total Páginas	37

Control de Versiones			
Versión	**Descripción de los cambios**	**Elaborado por**	**Fecha Elaboración**
v01r00	Elaboración inicial del documento	OT-I	26/04/2017
V01r01	Revisión y ajuste de formato	OT-I	20/11/20
v01r02	Ajuste de contenido	OT-I	25/11/20
v01r03	Actualización nueva plantilla	OT-I	28/01/21
v01r04	Actualización encabezamiento y pie a la nueva versión	OT-I	12/09/2024

Lista de Distribución	
Apellidos, Nombre	**Cargo / Función**

ÍNDICE

1. INTRODUCCIÓN....4

1.1. Objeto....4
1.2. Alcance....4
1.3. Condiciones de Uso....4

2. NORMAS Y BUENAS PRÁCTICAS....6

2.1. Elección entre SOAP y API RESTFul....6
2.2. Desarrollo de Servicios WEB....6
2.3. Desarrollo de servicios SOAP....7
2.4. Desarrollo de servicios RESTFul....9

3. CONCEPTOS GENERALES....11

3.1. Arquitectura Orientada a Servicios (SOA)....11
3.2. Perfil de Integración....12
3.3. Contrato del Servicio Web....13
3.4. Elementos Comunes....14
3.5. Comunicación SSL/TLS....15

4. SERVICIOS SOAP....16

4.1. Reglas de implementación para Servicios SOAP....16

5. SERVICIOS RESTFUL....27

5.1. Uso de los metodos HTTP....28
5.2. Reglas de implementación Servicios REST....32
5.3. Versionado....34

6. GESTIÓN DE ERRORES....36

7. BIBLIOGRAFIA Y REFERENCIAS....37

1. INTRODUCCIÓN

1.1. Objeto

El objetivo de este documento es establecer la normativa a seguir sobre la definición e implementación de Servicios Web SOAP y RESTFul que se desarrollen en la Consejería de Hacienda y Administración Pública.

1.2. Alcance

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.

- 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 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.

El **ámbito de aplicación es la Dirección General de Estrategia Digital y Gobierno Abierto de la Consejería de Presidencia, Administración Pública e Interior**, siendo una recomendación fuera de este contexto.

1.3. Condiciones de Uso

Las normas recogidas en esta guía son de **obligado cumplimiento**, dentro del alcance especificado. La Oficina Técnica de Interoperabilidad de la Consejería de Hacienda y Administración Pública (en adelante OT-I) se reserva el derecho a la modificación de la norma sin previo aviso, tras lo cual, notificará del cambio a los actores implicados para su adopción inmediata.

La OT-I podrá estudiar los casos excepcionales, en el caso de que algún actor considere necesario el incumplimiento de alguna de las normas y/o recomendaciones, deberá aportar previamente la correspondiente justificación fehacientemente documentada de la solución alternativa propuesta, así como toda aquella documentación que le sea requerida para proceder a su validación técnica.

Tras el análisis de la información aportada, la OT-I informará de manera explícita sobre las conclusiones obtenidas para lograr encontrar una solución adaptada en la medida de lo posible a las directrices marcadas.

2. NORMAS Y BUENAS PRÁCTICAS

A continuación se presenta un resumen de las normas y buenas prácticas de aplicación en el contexto del desarrollo.

Para más información, consulte los apartados que desarrollan los contenidos.

2.1. Elección entre SOAP y API RESTFul

La OT-I, de acuerdo a la tendencia tecnológica y a la evolución estratégica de la Dirección General recomienda la implementación de nuevos servicios web, API RESTFul, apoyándose en el estándar OpenAPI.

Sólo en aquellos casos en los que históricamente los servicio web de un sistema de información sean SOAP y el cambio tecnológico implique un esfuerzo desproporcionado con los beneficios del cambios, se mantendrán los desarrollos de servicios SOAP.

Para facilitar la toma de decisiones, y entendiendo que hay un sin fin de situaciones diferentes, la OT-I aportará soporte y asesoramiento experto en aquellos sistemas que lo precisen para diseñar la mejor solución de interoperabilidad que cubra todas sus necesidades.

2.2. Desarrollo de Servicios WEB

2.2.1. Elaborar un Perfil de Integración

Cuando se produce una integración entre **dos o más sistemas de información** entre los cuales se realizará un intercambio de información a través de **varios servicios web** se debe **crear un perfil de integración** para poder recoger todos los aspectos semánticos, sintácticos y técnicos que conlleve la integración.

Este documento será la base para las futuras evoluciones e incluso resolución de problemáticas que puedan surgir.

La OT-I pone a disposición de la Dirección General, en su grupo de RedProfesional, la plantilla sobre la cuál deberá realizarse el perfil: DGPD_NORM_Plantilla Perfil de Integración.

Para más detalle, consultar el apartado “Elaborar un Perfil de Integración”.

2.2.2. Adoptar una estrategia Contract-first

Se debe adoptar una estrategia de desarrollo Contract-first: primero definición del servicio y luego desarrollos.

Para más detalle, consultar el apartado "Adoptar una estrategia Contract-first".

2.2.3. Utilizar protocolos de comunicación SSL

La comunicación entre sistemas de información, cuando uno de ellos pertenece a la Dirección General debe ser a través del protocolo de comunicación SSL/TLS.

Para más detalle, consultar el apartado "Utilizar protocolos de comunicación SSL".

2.2.4. Seguir la política de versionado de servicios

Se deberá versionar el servicio ante cambios que impliquen la adaptación de los consumidores y mantener la versión anterior durante el tiempo acordado.

Para más detalle, consultar los apartados "Versionado de servicios" y "Versionado".

2.3. Desarrollo de servicios SOAP.

2.3.1. Documentar de forma diferenciada los elementos comunes

Para aquel sistema cuyo especificación disponga de elementos comunes a varios servicios web deberán recogerse de forma única en un documento de elementos comunes.

La OT-I pone a disposición de la Dirección General, en su grupo de RedProfesional, la plantilla sobre la cuál deberá realizarse el documento: DGPD_NORM_Plantilla Elementos Comunes.

Para más detalle consultar el aparatado "Elementos Comunes".

2.3.2. Publicar el WSDL

Los servicios web desarrollados deben hacer público su WSDL. Para más detalle , consultar el apartado "WSDL".

2.3.3. Emplear el patrón de diseño XML Schema

Los esquemas XML utilizados deberán cumplir las especificaciones WS-I Basic Profile. Para más detalle, consultar el apartado "Patrón de diseño XML Schema".

2.3.4. No utilizar XML incrustado en el propio mensaje

No se permiten parámetros que contengan XML incrustado como texto, todas las etiquetas han de estar definidas como parámetros. Para más detalle, consultar el apartado "Definición de atributos y operaciones".

2.3.5. Evitar el intercambio de documentos en los mensajes

Se recomienda evitar en la medida de lo posible la transmisión de ficheros sobre el body y usar, en su lugar, mecanismos de transmisión como MTOM. Para más detalle, consultar el apartado "Intercambio de documentos a través de servicios".

2.3.6. Aplicar la nomenclatura de los servicios

El nombre del servicio debe ser acorde a la funcionalidad del mismo con estilo CamelCase y solo con caracteres alfanuméricos. Para más detalle, consultar el apartado "Nomenclatura"

2.3.7. Aplicar la especificación de los servicios seleccionada

Se deben desarrollar los servicios acordes a la especificación elegida. Si el servicio es SOAP 1.1, todos los mensajes intercambiados deben seguir la especificación de SOAP 1.1. Para más detalle, consultar el apartado "Versionado de especificación.

2.3.8. Equilibrar la granularidad de operaciones por servicios

Se recomienda buscar un equilibrio entre el número de servicios y operaciones, evitando la defición del WSDL del servicio con una excesiva granularidad de operaciones. Para más detalle, consultar los apartados "Equilibrio entre servicios y operaciones" y "Granularidad gruesa".

2.3.9. Emplear la cabecera Conten-Type

Se debe usar la cabecera Content-Type "application/soap" o "apllication/soap+xml". Para más detalle, consultar el apartado "Context-Type".

2.4. Desarrollo de servicios RESTFul

2.4.1. Aplicar el estándar OpenAPI en el diseño de los servicios

Para la definición de una APIRest deberá hacerse uso del estándar de especificación OpenAPI

2.4.2. Utilizar correctamente los métodos HTTP

Se deben utilizar correctamente los métodos HTTP en función de la acción a realizar:

Operación	Método HTTP
Listar	GET
Crear	POST
Leer	GET
Actualizar	PATCH / PUT
Borrar	DELETE

Para más información, consulte el apartado correspondiente "Uso de los metodos HTTP"

2.4.3. Definición de URIs

Las URIs definidas deben seguir una serie de reglas descritas en el apartado "Nomenclatura URIs" .

3. CONCEPTOS GENERALES

3.1. Arquitectura Orientada a Servicios (SOA)

La evolución que los Sistemas de Información han experimentado en los últimos años han incrementado las necesidades de compartir información e interoperar entre ellos. Ante este escenario la OT-I ha definido una estrategia de interoperabilidad basada en una SOA como paradigma de arquitectura para sistemas distribuidos que cubra las necesidades funcionales de manera ágil, fiable y flexible, ganando además en seguridad y trazabilidad.

Para implementar con éxito una SOA se consideran destacables los siguientes principios:

- 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.
- Reutilización: Un servicio debe ser diseñado y construido pensando en su reutilización dentro de la misma aplicación o en la organización para su uso masivo.
- 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 compartir información y colaborar.
- Gobernabilidad: En un entorno SOA, la gobernanza guía el desarrollo de servicios reutilizables, estableciendo cómo se diseñarán, cómo se desarrollarán los servicios y cómo cambiarán con el tiempo. La Gobernanza establece acuerdos entre los proveedores de servicios y los consumidores de esos servicios, indicando a los consumidores lo que deben esperar y lo que los proveedores están obligados a proporcionar.

La pieza clave de una Arquitectura SOA son los servicios. Un Servicio Web es una parte del software que puede comunicarse con otra aplicación a través de una red usando un conjunto específico de protocolos estandarizados tales como SOAP, REST, 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.

Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los Servicios Web para hacer uso de los servicios expuestos en redes de ordenadores como Internet. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los Servicios Web.

Las operaciones que ofrezca un servicio 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. Esto permite la clara identificación de los servicios de negocio, disponiendo cada servicio de una responsabilidad única.

Los Servicios Web que se diseñen implicarán responsabilidades bien repartidas, de tal manera que sean escalables y reutilizables.

3.2. Perfil de Integración

Cada vez que exista una necesidad de integración, con mas de un servicio, que implique definir la interacción entre los distintos servicios debe crearse un perfil de integración para ello. En este documento se explica la solución de integración diseñada para una necesidad concreta. La plantilla que seguirá este documento es DGPD_NORM_Plantilla Perfil de Integración, en su última versión, y su estructura es:

- Introducción: En este apartado se describe el objeto del documento, que será la explicación sobre la necesidad de integración que se precise.

- Alcance del proceso de interoperabilidad: Descripción de las funcionalidades que debe cubrir el proceso de interoperabilidad, tales como contexto y requisitos.
- Descripción general del proceso de interoperabilidad: En este apartado se especifica los actores y los distinto 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 especifican 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.

  El Perfil de Integración debe ser el primer documento a leer por los desarrolladores de los servicios. Una vez se tenga el conocimiento funcional global de la integración se aumentará el conocimiento con la información técnica proporcionada en cada uno de los contratos de los servicios, referenciados en el propio documento del perfil.

3.3. Contrato del Servicio Web

Cada Servicio Web desarrollado ha de disponer de un contrato del Servicio asociado. Este contrato seguirá la plantilla DGPD_NORM_Plantilla Contrato Servicio Web, en su última versión, que se estructura de la siguiente forma:

- Servicio: En este apartado se describe los principales aspectos del servicio: nombre, tipología, objetivo, 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), encodig, mensaje de entrada y salida, junto con sus respectivos esquemas y parámetros de entrada/salida.
- Tipos de Datos: Descripción de los tipos de datos específicos del servicio que dan soporte a las operaciones del mismo.

Especificar que aquellos tipos de datos comunes para más de un servicio dentro del mismo Sistema vendrán informados en un documento aparte, cuya plantilla es DGPD_NORM_Plantilla Elementos Comunes, en su última versión.

- Tablas de referencia: Descripción de la información, específica del servicio, que por razones de diseño se intercambie en el servicio mediante su código correspondiente.

  Especificar que aquellas tablas de referencias comunes para más de un servicio dentro del mismo Sistema vendrán informados en el documento de elementos comunes, cuya plantilla es DGPD_NORM_Plantilla Elementos Comunes, en su última versión.
- Gestión de errores: Para cada uno de los error 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 como anexos, glosario y referencias.

Adicionalmente a la cumplimentación de este documento se ha de adjuntar el WSDL en servicios SOAP, y los esquemas de datos utilizados (JSON o XSD).

3.4. Elementos Comunes

Cada sistema deberá disponer de un documento que recoja los elementos comunes para todos los sistemas web que publique. 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 que se generó a partir de la plantilla DGPD_NORM_Plantilla Elementos Comunes, en su última versión, y que tiene la siguiente estructura:

- Tipos de datos: Recoge aquellos datos estructurados comunes a varios servicios Web del Sistema en cuestión.
- Tablas maestras: Recoge las tablas maestras del sistema que son compartidas por varios servicios web.

- Gestión errores: muestra los errores que pueden darse de manera genérica en varios servicios del Sistema.

3.5. Comunicación SSL/TLS

Se establece el uso de SSL/TLS como protocolo de comunicación para aquellos servicios que se consuman o se ofrezcan tanto en sistemas ubicados dentro de la CHAP como los ubicados fuera de la misma. Este protocolo facilita la comunicación cifrada permitiendo la confidencialidad del dato/mensaje intercambiado entre dos sistemas. De esta forma, se proporciona seguridad ante aquellos accesos a servicios que se realicen desde cualquier punto con acceso a internet (cafeterías, aeropuertos, …) cuyas conexiones no siempre encriptan las comunicaciones y resultan fácilmente espiados o falsificados si son interceptadas las credenciales de autenticación.

4. SERVICIOS SOAP

SOAP (siglas de Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es un paradigma de mensajería de una dirección sin estado, que puede ser utilizado para formar protocolos más complejos y completos según las necesidades de las aplicaciones que lo implementan. Por ello, se definirán como servicios SOAP aquellos servicios que necesiten un mecanismo de seguridad complejo.

4.1. Reglas de implementación para Servicios SOAP

4.1.1. WSDL

WSDL (Web Services Description Language) es un XML que se utiliza para describir la interfaz pública de los Servicios Web de forma estandarizada.

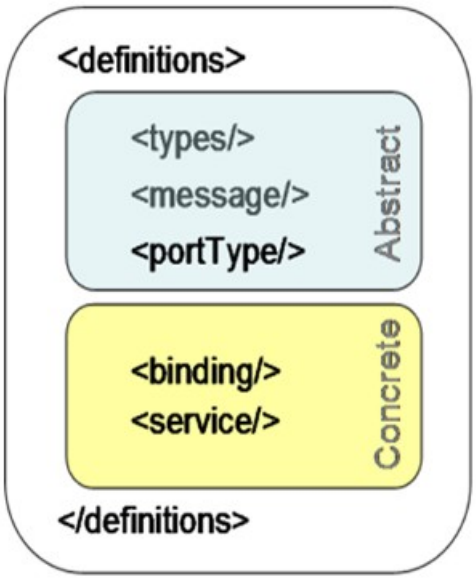
4.1.1.1. Definición WSDL

Un documento WSDL describe tres propiedades fundamentales de un servicio web:

- Las operaciones soportadas y qué mensajes las activan.
  - El formato de los mensajes.
  - Los tipos de datos especiales que se envíen se incluyen en el archivo WSDL en forma de XML Schema.
- El protocolo de comunicación en el que se envía el mensaje.
- La forma en qué cada operación se compone de mensajes formateados de una forma específica y transmitidos por un protocolo concreto de red

Un WSDL dispone de dos partes; la parte abstracta y la parte concreta:

- En la parte abstracta se incluye:
  - Types: Esta etiqueta define las estructuras de datos que se utilizarán para construir los mensajes de petición como de respuesta. Estas estructuras de datos pueden construirse con cualquier lenguaje, pero lo más normal es hacerlo con XML Schema.
  - Message: Describe los mensajes que se van a intercambiar entre el cliente y el Servicio Web.



Un mensaje puede estar dividido en varias partes, por ejemplo, si en un mensaje queremos enviar datos y una imagen.
    - PortType: Define el conjunto de operaciones que soporta el Servicio Web. Una operación no es más que un grupo de mensajes que serán intercambiados. Cada operación puede enviar o recibir al menos un mensaje cada vez.
- En la parte concreta se incluye:
    - Binding: Describe como formatear los mensajes para interactuar con un Servicio determinado. WSDL no define un estándar para formatear mensajes. Para ello utilizar la extensibilidad para definir como intercambiar los mensajes usando SOAP, HTTP, MIME, etc…
    - Services: Este elemento indica donde se encuentra el Servicio usando la etiqueta. Cada etiqueta define el formato de los mensajes, y la dirección donde se encuentra el servicio que acepta mensajes en ese formato.

Cada Servicio Web ha de disponer y publicar el WSDL que han de ser auto documentados, es decir han 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.

4.1.1.2. Namespaces

La definición de los namespaces dentro de los Servicios Web es un tema muy importante a tener en cuenta. Una correcta definición de los mismos favorece la reutilización y catalogación de los servicios y posibilita la creación de cierto tipo de servicios horizontales de enrutación y mediación. Se ha detectado que los ficheros de XSLT parecen tener un contador interno y una vez se supera este contador, acaba arrojando un error 500. Para evitar este error, como buenas prácticas en la creación de namespaces, se deberá hacer una correcta definición de los mismos, evitando la generación dinámica de nodos en el esquema.
Los namespaces dentro de los WSDLs y esquemas empleados dentro de los Servicios Web, se normalizan dentro de MADEJA. 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. El formato recomendado por MADEJA y el obligatorio en los desarrollos en CHAP es el de urn, al ser más claro y requerir menos procesos de normalización que el de URLs.
El formato y la descripción de los campos es el presentado a continuación.

Formato		
urn:juntadeandalucia:chap:(aplicación):(versión):(servicio):(componente -opc)		
Descripción de los campos.		
Elemento	Obligatorio	Descripción
Juntadeandalucia	Si	Referencia a la Junta de Andalucía
chap	Si	Referencia a la Consejería de Hacienda y Administración Pública
Aplicación	Si	Nombre de la aplicación a la que pertenece el servicio. Se usará el código de aplicación asignado en QM. Será obligatorio para todos los casos, excepto para tipos que se consideren común a toda la Consejería.
Versión	Si	Número de versión
Servicio	No	Nombre del servicio.
Componente	No	Nombre del elemento o componente

4.1.2. Definición de servicios (Contract-First)

A la hora de desarrollar un Servicio Web el programador puede optar por desarrollarlos siguiendo la aproximación "Code-First" o la aproximación "Contract-First".
En la aproximación "Code-First", primero se codifica el código del servicio dentro del lenguaje de programación y después se genera el WSDL describiendo el servicio de forma automática, a través del framework de Servicios Web elegido. Esta opción presenta una serie de inconvenientes que desaconsejan su uso. El contrato del servicio está muy ligado a la implementación del mismo, de forma que un cambio en dicha implementación puede conllevar cambios en el WSDL. Además, el contrato está también muy ligado al framework de programación empleado, de forma que un cambio de framework o una actualización de versión del mismo, puede conllevar cambios en el WSDL, con las consiguientes implicaciones en los consumidores de dicho servicio.

En la opción "Contract-First", la idea es la contraria, primero se genera el WSDL y después se codifica el servicio. Esta opción carece de los inconvenientes de la opción anterior y aporta numerosas ventajas a la hora de desarrollar y mantener los Servicios Web. En esta opción hay que 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 Servicios Web. 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 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.
Por todo ello, es obligatorio que los Servicios Web desarrollados sigan la aproximación Contract-first.

4.1.3. 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

4.1.4. 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.

Este tipo de prácticas 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.
A nivel de orquestación en la Plataforma de Interoperabilidad, este tipo de prácticas complica el tratamiento de los mensajes enviados, así como la validación del mismo contra esquemas.
Para evitar esta problemática 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.

4.1.5. Intercambio de documentos a través de servicios

La trasmisión de ficheros mediante Servicios Web se debe realizar de una forma optimizada, o bien haciendo uso de MTOM o bien como attachments.
MTOM es un mecanismo de optimización que proporciona una forma efectiva y normalizada (por W3C) de transmitir ficheros, donde en lugar de proporcionar el contenido en el body codificado en Base64, envía los datos binarios como adjunto.
Debe evitarse, en al medida de lo posible, la transmisión de ficheros sobre el body.

4.1.6. Nomenclatura

A continuación se describen las reglas de nomenclatura para el desarrollo de servicios.

- El nombre del servicio debe describir lo mejor posible la funcionalidad. De esta forma, los desarrolladores y administradores de sistemas pueden predecir/intuir fácilmente la función del servicio basándose en su nombre.
- Hay que tener en cuenta que se distingue entre mayúsculas y minúsculas.
- Con respecto al nombre del servicio, debe utilizarse el estilo "CamelCase", el cual es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra en CamelCase se asemejan a las jorobas de un camello.
  Existen dos tipos de CamelCase:
  - UpperCamelCase, cuando la primera letra de cada una de las palabras es mayúscula. Ejemplo: EjemploDeUpperCamelCase.
  - lowerCamelCase, igual que la anterior con la excepción de que la primera letra es minúscula. Ejemplo: ejemploDeLowerCamelCase. Este estilo es el que se ha de usar.
    Ejemplo: consultaDatosUsuario en lugar de consulta_datos_usuario
- Utilizar sólo caracteres alfanuméricos (no utilizar los caracteres .; : | _ ,).

4.1.7. Versionado de servicios

Los Servicios Webs no son elementos inmutables en el tiempo, sino que están sujetos a cambios a lo largo de toda su vida. El componente distribuido de los Servicios Web hace que los cambios en su interfaz puedan repercutir en los consumidores, obligándoles incluso a volver a implementar su interfaz con el servicio.

Todo esto desemboca en la necesidad de definir una política de versionado de los Servicios Web, que aborde la creación de reglas que regulen la evolución de los mismos. Para ello se debe identificar los tipos de cambios que se realizan en un servicio y tener en cuenta el impacto de dichos cambios.

A continuación se detallan los diversos tipos de cambios que se han establecido para un Servicio Web:

- 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.

Como se ha comentado en el apartado de nomenclatura, el nombre del servicio incluye el número de versión, el cual no variará ante los cambios con compatibilidad hacia atrás (cambios menores o revisión), y que se incrementaría en caso contrario.

Ejemplo: v01/nombreServicio v02/nombreServicio

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. La OT-I garantizará el acceso al servicio anterior durante un plazo máximo de dos meses, tiempo en el que el cliente del servicio debe tener desarrollado la nueva versión del servicio.
- Los cambios de versiones de los servicios han de ser notificados a los clientes, previa actualización del WSDL y del contrato del servicio. En casos excepcionales en los que sea necesario mantener mas de dos versiones en paralelo de un servicio debido a que un cliente no pueda adaptarse, se trasladará a la OT-I.

Esta misma política de versionado se ha de seguir para el versionado de esquemas de datos.

4.1.7.1. Versión menor

Son pequeñas modificaciones que afectan al contrato del Servicio pero son compatibles 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>
```

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>
```

4.1.7.2. Versión mayor

Son modificaciones incompatibles con el contrato anterior. Es un cambio que si afecta a la compatibilidad hacia atrás del servicio, por lo que se deberá incrementar el número de versión del nombre del servicio.

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 el tiempo que se acuerde con la OT-I. Durante este tiempo se adaptarán los consumidores del servicio a la nueva versión.

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.

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>
```

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>
```

1.

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>
```

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>
```

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>
```

4.1.8. Versionado de especificación

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.

Por lo que será necesario desarrollar utilizando siempre la misma versión de especificación o versiones compatibles.

4.1.9. 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.

4.1.10. Granularidad gruesa

A la hora de diseñar las operaciones, es más eficiente utilizar un único mensaje de gran tamaño, que su equivalente en múltiples mensajes. Uno de los errores habituales cuando se trabaja con WSDL es definir operaciones con excesiva granularidad. Es mejor definir servicios de grano-grueso y que los servicios estén orientados al negocio mas que a las interfases de programación. No se debe definir una operación de servicio por cada método Java que se quiera exponer. Siempre hay que pensar en las necesidades desde la perspectiva del negocio mas que en las técnicas a la hora de exponer los servicios.

4.1.11. Context-Type

El campo Context-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 ContextType es "text/xml". Sin embargo, no está soportado en SOAP 1.2, por lo que el valor que debe configurarse para el desarrollo de Servicios Web es "application/soap" o "application/soap+xml".

5. SERVICIOS RESTFUL

Los servicios REST se desarrollaran cuando el mecanismo de seguridad que deba aplicarse sea poco restrictivo.

Una API es la abreviatura de "Interfaz de Programación de Aplicaciones" (Aplication Programming Interface) cuyo objetivo es proporcionar unos servicios de negocio útiles, aunque con un matiz importante, está orientada a hacer la vida más fácil al desarrollador.

Una API REST es una interfaz entre sistemas que usa HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa a SOAP por su menor complejidad y por la facilidad que proporciona en la manipulación de datos.

La arquitectura REST (Representational State Transfer) se define 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.

Se usa, en el sentido más amplio, para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc.) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP.

Los servicios REST se caracterizan por los siguientes fundamentos:

- 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. Ventajas que ofrece REST para el desarrollo:

- Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
- Visibilidad, fiabilidad y escalabilidad. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.
- La API REST siempre es independiente del tipo de plataformas o lenguajes: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

5.1. Uso de los metodos HTTP

Es importante conocer el significado o la utilidad de cada verbo HTTP para hacer un uso correcto del mismo. Únicamente está permitido el uso de los verbo GET, POST, PUT y DELETE:

5.1.1. GET

Se usa para obtener información del servidor, puede ser algún archivo HTML, una imagen, un archivo de texto, un XML, etc. Este verbo solo debe usarse para obtener información del servidor de acuerdo a los estándares de HTTP. El método GET no debe cambiar el estado del servidor, es decir, no debe hacer ninguna modificación a ningún recurso del servidor.

Debe usarse este verbo, por tanto, para realizar consultas sobre recursos. Dado que la ejecución de este verbo no modifica estado en el servidor, es recomendable su uso para el cacheo de datos.

Ej.: Obtener listado de elementos

GET /api/v1.0/elementos/

Ej.: Obtener elemento concreto

GET /api/v1.0/elementos/{idElemento}

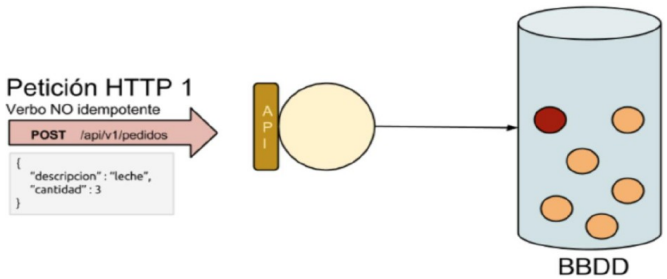
5.1.2. POST

Este suele ser el método HTTP más empleado, es el encargado de crear un nuevo recurso y, por consiguiente, modifica el estado del servidor.

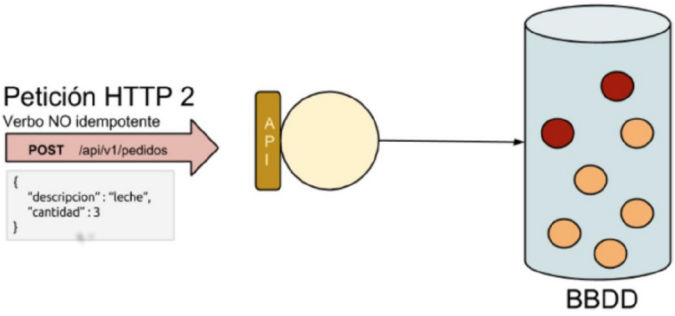
El método POST se confunde en ocasiones con el método PUT, aunque la diferencia radica en una cualidad que se llama idempotencia. La idempotencia se trata de la capacidad que tenga un ente (en este caso el método) de realizar una misma operación, con los mismos parámetros y valores, varias veces, y obtener siempre el mismo valor de salida.

Se muestra a continuación en el caso de peticiones POST el comportamiento esperado del sistema

1. Se envía una petición de creación de un recurso con unos parámetros: El estado en el sistema se modifica, creándose un nuevo pedido en base de datos.



2. Se envía una petición de creación de un recurso con unos parámetros: El estado en el sistema se modifica, creándose un nuevo pedido en base de datos.



De esta forma, tal y como muestra el ejemplo, el comportamiento esperado del verbo POST sobre un recurso se traduce siempre en un cambio del estado del mismo en el sistema, aunque se apliquen los mismos parámetros sobre el mismo recurso.

La utilización de este verbo debe limitarse al alta de nuevas entidades.

Ej.: Crear un nuevo elemento

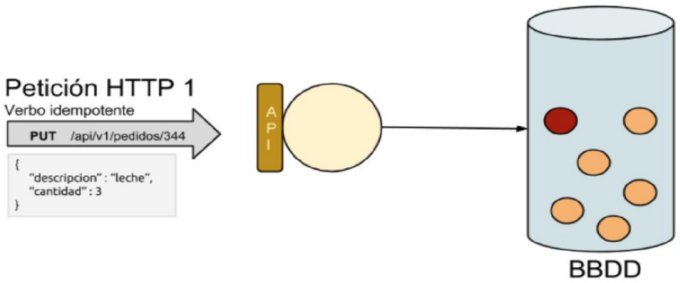
```
POST /api/v1.0/elementos/
Content-Type: application/json
Accept: application/json
{
    "idElemento": 123,
    "dato1": "Dato 1",
    "dato2": "Dato 2"
}
```

5.1.3. PUT

No es tan usado como lo son GET y POST. Este verbo también implica una acción sobre el estado del servidor pero se desea que la siguiente vez que suceda, con los mismos parámetros y valores, no cambie ese estado. Teniendo esto en cuenta, podemos relacionar el método PUT con la acción de actualizar.

A diferencia del verbo POST, PUT si es un verbo idempotente. La ejecución repetida de una petición con PUT con los mismos parámetros sobre un mismo recurso tendrá el mismo efecto en el estado del recurso en el sistema si se ejecuta 1 o N veces. Se muestra a continuación un ejemplo:

1. Se envía una petición de creación/actualización sobre un recurso con unos determinados parámetros: Se crea un nuevo recurso en el sistema en caso de no existir, o se modifica en caso de que algo haya cambiado sobre uno existente.



2. Se repite la petición sobre el mismo recurso y con los mismos parámetros: Al tratarse de una actualización sobre un recurso ya existente, el estado del sistema se mantiene igual. No se crea un recurso nuevo y el valor de las propiedades del recurso es el mismo que cuando finalizó la primera petición.

PUT debe usarse este verbo para modificación de recursos ya existentes.

Ej.: Actualizar un nuevo elemento

```
PUT /api/v1.0/elementos/{idElemento}
Content-Type: application/json
Accept: application/json
{
    "dato1": "Dato 1 Actualizado",
    "dato2": "Dato 2 Actualizado"
}
```

5.1.4. DELETE

El verbo DELETE es el único que debe ser usado para borrar un recurso del servidor, no estando permitido borrar recursos con los verbos GET o POST. Este verbo también tiene la cualidad de idempotencia.

Ej.: Eliminar un elemento

```
DELETE /api/v1.0/elementos/{idElemento}
```

5.2. Reglas de implementación Servicios REST

5.2.1. Nomenclatura URIs

Los recursos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.

Las URIs han de seguir una serie de reglas:

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

  ../api/v01/usuarios/{dni}/informes/tipoInforme/{idInforme}

- Nomenclatura de campos y parámetros CamelCase (lowerCamelCase)
- Los nombres de URI no deben implicar una acción, por lo tanto no se podrán usar verbos en ellos.
- Han ser únicas, no se puede haber más de una URI para identificar un mismo recurso.
- Han de incluir la versión de la API.

  ../api/v01/elementos/

- Han 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 vayamos 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/cliente/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 en lugar de incluirlos en la misma.

    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 utilizaríamos 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 permitirá 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 símbolo "-" 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. Esta información es útil puesto que se establece como norma que se devuelva la información funcional mínima requerida en cada consulta, de manera que si esta fuera necesaria ampliarla se pueda concatenar campos mediante este parámetro (separados por ","). Se puede indicar el valor "all" a este parámetro de modo que se han de añadir todos los campos a la respuesta.

5.3. Versionado

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 no es necesario liberar una nueva versión, basta con reemplazar la actual. Será obligatorio informar a los consumidores de los Servicios REST de los nuevos cambios para que los tengan en cuenta.

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

- Añadir nuevas operaciones al servicio. Traducido a REST sería añadir nuevas acciones sobre un recurso (PUT, POST…)
- Añadir parámetros de entrada opcionales a peticiones sobre recursos ya existentes. Ejemplo: un nuevo parámetro de filtrado en un GET sobre una colección de recursos.
- Modificar parámetros de entrada de obligatorios a opcionales. Ejemplo: al crear un recurso, una propiedad de dicho recurso que antes fuese obligatoria y que pase a opcional.
- Añadir nuevas propiedades en la representación de un recurso que nos 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 si afectan a los consumidores de los servicios y que obligan a versionar el 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 el tiempo que se acuerde con la OT-I. 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 si obligan a versionar la API:

- Eliminar operaciones o acciones sobre un recurso. Ejemplo: Ya no se aceptan peticiones POST sobre un recurso.
- Añadir nuevos parámetros de entrada obligatorios. Ejemplo:Ahora para dar de alta un recurso hay que enviar en el cuerpo de la petición un nuevo campo requerido.
- Modificar parámetros de entrada de opcional a obligatorio. Ejemplo: Ahora al crear un recurso Persona, el campo edad, que antes era opcional, ahora es obligatorio.
- Modificar un parámetro en operaciones (verbos sobre recursos) ya existentes. También aplicable a la eliminación de parámetros. Ejemplo: al consultar un recurso ya no se devuelve determinado campo. Otro ejemplo: un campo que antes era una cadena de caracteres, ahora es numérico.
- Añadir nuevas respuestas en operaciones ya existentes. Ejemplo: ahora la creación de un recurso puede devolver un código de respuesta 412.

La versión se incluye en la propia URI:

../api/v01/usuarios/{idUsuario}/Informes/tipoInforme/{idInforme}

6. GESTIÓN DE ERRORES

La normativa relativa a la gestión de errores de los Servicios Web está recogida en el documento DGPD_NORM_Politica de Gestion de errores, en su última versión.

7. BIBLIOGRAFIA Y REFERENCIAS

Referencia	Fuente
Buenas Prácticas para diseñar una API RESTful	https://unpocodejava.wordpress.com/2013/11/04/buenas-practicas-para-disenar-una-api-restful-pragmatic-parte-i/
SOAP	https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol https://www.w3.org/TR/2003/REC-soap12-part0-20030624/
REST	https://es.wikipedia.org/wiki/Representational_State_Transfer
Basic Profile 1.1	http://www.ws-i.org/Profiles/BasicProfile-1.1.html
Basic Profile 2.0	http://ws-i.org/Profiles/BasicProfile-2.0-2010-11-09.html
Web Services Interoperability Organization	http://www.ws-i.org/deliverables/workinggroup.aspx?wg=testingtools