DWES06 - SERVICIOS WEB

Tipo de servicios web

Según el W3C existen dos grandes tipos de servicios web:

• Big Web services: servicios que exponen operaciones para que las utilicen los servidores. Están basados en el protocolo SOAP.

• Web API: servicios basados en la arquitectura REST -que basa su funcionamiento en los recursos-.

Big Web services

Los conocidos como big Web services son servicios web que envían y reciben la información empaquetada en un formato XML llamado SOAP. Se pueden definir a partir de tres funciones, aunque no todas son estrictamente necesarias para trabajar:

• Una forma de definir y transmitir los mensajes. • Un
lenguaje de definición de cómo se pueden utilizar los servicios. • Un sistema de descubrimiento de servicios.

Formato de los mensajes

Para definir el formato de los mensajes que se transmiten se utiliza el protocolo SOAP, basado en XML.

Para transportar estos mensajes suele utilizarse HTTP aunque se pueden utilizar otros protocolos de transporte como los de correo electrónico, FTP.

Definición del servicio

Para poder utilizar un servicio web que nunca se ha utilizado y que incluso puede haber sido descubierto automáticamente es muy importante disponer de una manera de conocer qué servicios ofrece y cómo debe hacerse para usarlos.

La forma que tienen estos servicios web de definir cómo hacer para poder usar el servicio y qué servicios se ofrecen es mediante otro protocolo llamado WSDL (Web services description language), que también está basado en XML.

Descubrimiento de servicios

Si se pretende que los programas funcionen sólo es necesario que tengan alguna manera de poder descubrir servicios que no conocían anteriormente de forma automática.

Se definió UDDI (descripción, descubrimiento e integración universales o universal description, discovery and integration) para proporcionar la capacidad de permitir a los programas descubrir servicios web que se adecuen a sus necesidades.

SOAP

SOAP fue el primer estándar de servicios web que se desarrolló y en la actualidad es una recomendación del W3C. Es un acrónimo de simple object access protocol (protocolo de acceso de objeto simple), pero actualmente este nombre ha perdido sentido puesto que ha quedado demostrado que SOAP no es simple, y además no tiene mucho que ver con el acceso a objetos .

Han ido surgiendo diferentes versiones de SOAP a lo largo de los años y actualmente se utilizan dos, las versiones 1.1 y 1.2, cuya especificación se puede encontrar en la página del W3C (http://www.w3.org /TR/soap/).

Es un entorno de mensajes que permite la comunicación entre programas que corren en distintos servidores con independencia del lenguaje y del sistema operativo en el que funcionen.

Consigue la independencia del lenguaje y de la plataforma basando los mensajes en XML y se aprovecha que el éxito de la web ha hecho que todos los sistemas tengan implementado el protocolo HTTP (aunque puede funcionar en otros protocolos) para hacerlo servir para el intercambio de mensajes.

Además de definir el formato de los mensajes también define cuáles son las reglas que deben regir este intercambio, qué ocurre en caso de error y cómo debe ligarse con los demás protocolos.

Con los servicios web de este tipo se exponen toda una serie de funciones mediante las cuales se pueden realizar programas que pueden hacer que el servicio realice tareas determinadas y devuelva resultados con los que se pueda trabajar.

Composición de los mensajes

Un mensaje SOAP es básicamente un documento XML y por tanto debe cumplir todas las normas de creación de documentos XML.

Problemas

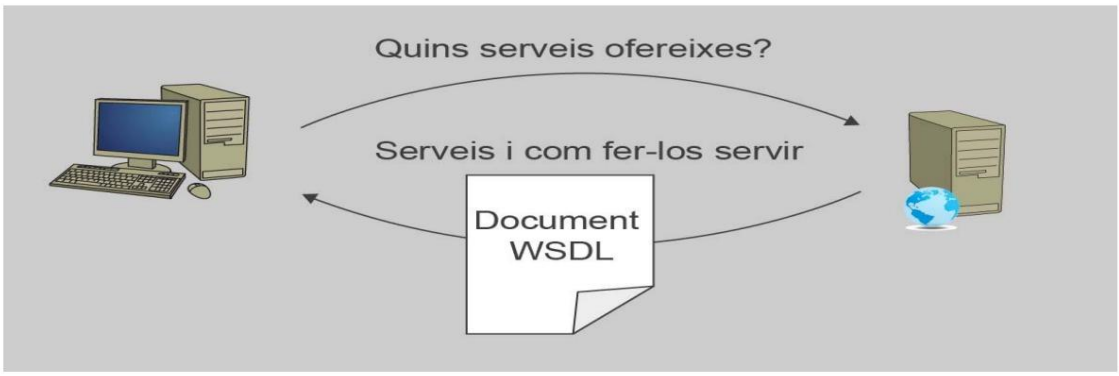
Una de las críticas que se hace en SOAP es la misma que en XML: a pesar de tener muchas ventajas tiene la tendencia a hacerse muy grande y esto hace que SOAP acabe siendo más lento que las otras tecnologías existentes.

Aunque estos problemas son ciertos, para mucha gente las ventajas que aporta -como la facilidad de lectura para los humanos, la facilidad de detección de errores y el hecho de que solucione los problemas de compatibilidad entre plataformas- lo hacen una opción debe tener en cuenta.

WSDL

Web services definition language (WSDL) es un lenguaje XML para describir qué servicios ofrece un servicio web y cómo hacerlo para acceder a él.
Normalmente el uso de un servicio web requiere que el cliente obtenga antes el archivo WSDL para saber qué servicio se ofrece y cómo utilizarlo.

WSDL sirve para obtener las características técnicas de un servicio web



Desarrollo de servicios web

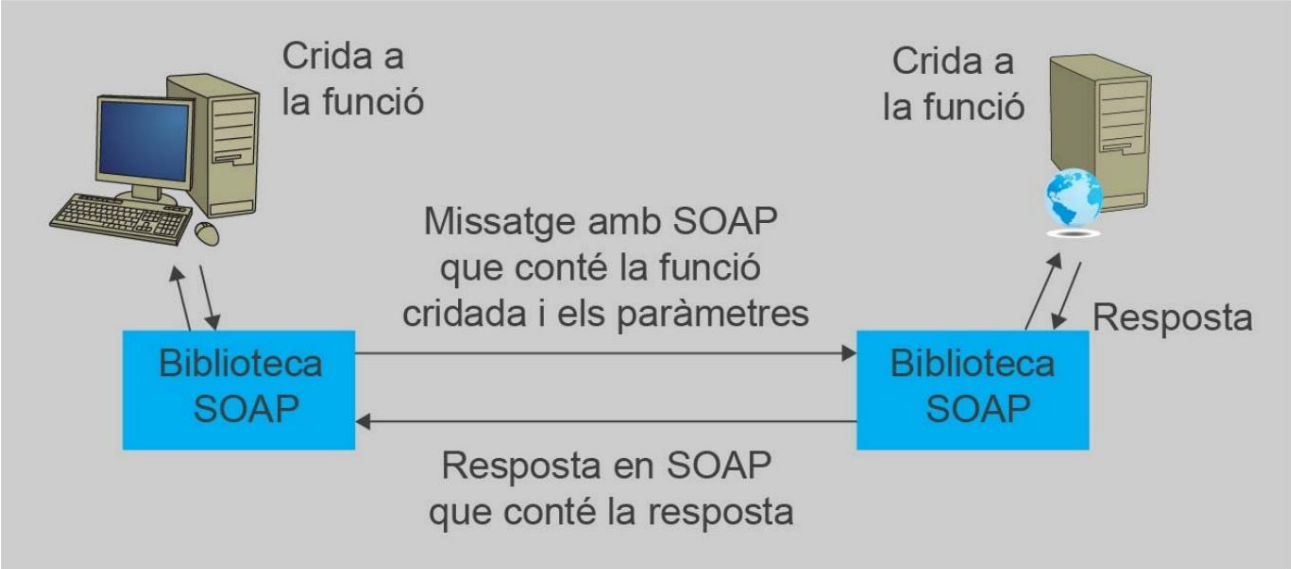
Las ventajas que pueden aportar los servicios web (web services) no han pasado desapercibidas y rápidamente los lenguajes de programación han ido incorporando componentes para desarrollarlos. Por ejemplo, Java development kit (JDK) de Java incorpora una API para crear servicios web llamado Java Api for XML Web Services (JAX-WS).

La mayoría de las bibliotecas o entornos de desarrollo se especializan en uno de los dos grandes grupos de servicios web, pero el incremento de la popularidad de los servicios REST ha hecho que muchos de los entornos basados en el protocolo SOAP también incorporen la posibilidad de funcionar de forma compatible con REST.

Desarrollo de servicios basados en SOAP

Las bibliotecas SOAP están pensadas para ser transparentes de cara a los programadores, y por tanto estos programadores no suelen tener demasiados problemas para utilizarlas. Es así porque las librerías SOAP se encargan de crear y eliminar los mensajes SOAP, y en el programa sólo llegan los datos que se han enviado.

Un cliente solicita un servicio e internamente la biblioteca crea el paquete SOAP para los datos que debe enviar y se lo envía. Cuando la biblioteca recibe los datos elimina el paquete SOAP y envía los datos limpios al programa



Las bibliotecas SOAP esconden la implementación en los programadores

Por tanto, excepto en pequeños detalles, los programadores pueden despreocuparse de la existencia de SOAP y sólo tienen que preocuparse de qué servicios quieren ofrecer.

Ejemplo utilizando php.

Existe en php una clase para poder trabajar con SOAP llamada SoapClient. Se necesita conocer el wsdl del servicio que queremos utilizar. Por ejemplo, para conocer los resultados de la eurocopa de Francia: http://footballpool.dataaccess.eu/data/info.wso?wsdl <?php

```
        $client = new SoapClient("http://footballpool.dataaccess.eu/data/info.wso?wsdl");

        //print_r($client-> getFunctions()); // para descubrir las funciones disponibles

        $result = $cliente->AllGames();
        $array = $result->AllGamesResult->tGameInfo;

        echo "<table border='1'>";
        echo "<tr><th>Fecha</th><th> </th><th> </th><th>Resultado</th></tr>";

        foreach($array a $k=>$v)
                { echo "<tr>"; echo
                "<td align='right'>" . $v->dPlayDate . "</td>
                <td><img src='{$v->Team1->sCountryFlag}'>" . $v->Team1->sName . "</td>
                <td align='right'>" . $v->Team2->sName . "<img src='{$v->Team1->sCountryFlag}'></td>
                <td align='center'>" . $v->sScore . "</td>"; echo "</tr>";

        }

        echo "</table>";
?>
```

SERVICIOS REST

REST deriva de REpresentational State Transfer .

REST es cualquier interfaz entre sistemas que utiliza HTTP para obtener datos o generar operaciones sobre estos datos utilizando preferiblemente los formatos XML y JSON. Es una alternativa a los protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también de mucha complejidad.

Características de REST

- Protocolo cliente/servidor sin estado (stateless): Esto implica que el sistema debe tratar cada una de las peticiones como si fuera la única, y por tanto debe tener todo lo necesario para que el servidor pueda realizar la tarea. El servidor nunca tendrá en cuenta las peticiones anteriores para tratar el actual.

- Las operaciones más importantes relacionadas con los datos en un sistema REST son cuatro: POST (crear), GET (leer), PUT (editar), DELETE (borrar).

- Los objetos en REST siempre se manipulan a partir de la URI. Es la URI el identificador único de cada recurso.

- Interfaz uniforme: para la transferencia de datos en un sistema REST, realizamos acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, identificados con una URI.

- Sistema de capas: arquitectura jerárquica entre sus componentes. Cada una de estas capas implementa una determinada funcionalidad en el sistema REST.

- Utilización de hipermedia. Parte de la información de las respuestas serán hipervínculos de navegación asociados a otros recursos.

```
{ "id":
 78, "nombre":
 "Joan", "linaje":
 "García", "coches":
[ {"coche": "http://servidor/concesionario/ api/v1/clients/78/ coches/1033" },
{"coche": "http://servidor/concesionario/api/v1/clients/78/coches/7033"} }
```

Buenas prácticas utilizando REST

Utilizar los métodos HTTP para realizar las diferentes peticiones

Los métodos HTTP mejoran la claridad de las peticiones Los cuatro verbos más utilizados son los siguientes:

GET

Leer un determinado recurso (con un identificador) o una colección de recursos

PUT

Modificar un recurso específico o una colección de recursos. Puede utilizarse también para crear un determinado recurso si se conoce el identificador del recurso previamente.

DELETE

Borra un determinado recurso a partir de su identificador.

POST

Crear un recurso nuevo. Se utiliza para operaciones que no coinciden con ningún método.

Emplear nombres de recursos claros y apropiados.

Emplear nombres de recursos claros y apropiados nos ayuda a entender cuál será la respuesta de la petición.

Los nombres de recursos apropiados hacen que nuestra API sea mejor entender. Los recursos son tratados como una jerarquía a través de sus nombres URI.

Algunas reglas importantes a la hora de diseñar nuestros nombres de recursos:

- Utilizar identificadores en nuestras URLs en lugar de parámetros

  www.lloc.com/usuarios/12345
  en lugar de www.lloc.com/api?
  tipus=usuario&id=12345

- Emplear la estructura jerárquica de las URL para simplificar los nombres de los recursos.
- Diseñar pensando con los clientes y no con los datos. • Los nombres de recursos deben

  ser nombres. Evitar utilizar verbos para construir los nombres de recursos. Para especificar la parte correspondiente al verbo utilizaremos los métodos HTTP. • Utilizar los plurales

  www.lloc.com/clients/33245/comandos/8769/linies/1
  mejor que www.lloc.com/client/33245/comanda/8769/linia/
  1

  mejor utilizar clientes que cliente_lista.

- Utilizar minúsculas separando las palabras con • _ o -

  Realizar las URLs lo más corto posible.

Usar los códigos de respuesta HTTP para indicar estado.

Los códigos de respuesta son parte del protocolo HTTP. Podemos utilizarlos para comunicar el resultado de una petición. Por ejemplo si un recurso se crea correctamente a partir de una petición POST nuestra API puede devolver un 201 Algunos códigos de estado HTTP que se pueden utilizar son:

200 OK : Correcto. El más utilizado.

201 CREATED: Creación correcta (POST o PUT). Puede haber contenido o no dentro del cuerpo de la respuesta (response body)

204 NO CONTENTO : Correcta por las operaciones DELETE o PUT (no hay ningún contenido en el cuerpo de la respuesta).

400 BAD REQUEST : Error general.

401 UNAUTHORIZED : Error token de autentificación incorrecta.

403 FORBIDDEN: Error. Usuario no autorizado.

404 NOT FOUND: Recurso inexistente (a veces enmascara los códigos 401 o 403).

405 METHOD NOT ALLOWED : El método especificado no es correcto.

409 CONFLICTO: Por ejemplo borrado en cascada no soportado, entrada duplicada,...

500 INTERNAL SERVER ERROR: Error interno producido por una excepción.

Ofrecer JSON y XML

Normalmente se utiliza mucho más JSON. Lo ideal sería que los clientes pudieran elegir la respuesta con xml o json. La utilización de xml supone también la implementación de esquemas de validación y espacios de nombres y por tanto supone mucho más complejidad.

Evitar crear respuestas complejas.

Es mejor crear respuestas simples que implementen operaciones CRUD sobre los recursos simples que crear respuestas complejas.

Emplear los links hipermedia.

Si se utilizan links dentro de las respuestas las APIs son mejores de entender y utilizar.
Por ejemplo cuando se devuelve una colección de recursos proporcionar links para la paginación (primero, último, anterior, siguiente) es muy útil o utilizar HTTP Location header con el link a un recurso creado por PUT o POST.