Universidad de Vigo:
Metodologías para el Desarrollo de Servicios en la Web

Revisión de los Servicios Web SOAP/REST: Características y Rendimiento

Alberto Los Santos Aransay, albertolsa@gmail.com

Marzo - 2009

ÍNDICE

1. INTRODUCCIÓN 1
2. ARQUITECTURAS ORIENTADA A SERVICIOS 2
3. SERVICIOS WEB 6
3.1. SOAP WS 7
3.1.1. WS-* 11
3.2. Servicios Web REST 12
3.2.1. Arquitectura REST 13
3.2.2. Ejemplo 14
4. CARACTERÍSTICAS SOAP y REST 17
4.1. Debate SOAP vs REST 18
5. RENDIMIENTO 20
6. CONCLUSIONES 23
7. REFERENCIAS 24

ÍNDICE DE FIGURAS

Figura 1: Arquitectura orientada a servicios [2] 2
Figura 2: Los Servicios Web se basan en estándares abiertos para interconectar aplicaciones y usuarios independientemente de las plataformas [3] 6
Figura 3: Interacción a través de Servicios Web [9] 8
Figura 4: Estructura de los mensajes [10] 9
Figura 5: Los servicios Web en Funcionamiento [11] 10
Figura 6: Ejemplo con las funciones de REST [20] 13
Figura 7: Google Apps [29] 19

1. INTRODUCCIÓN

Coulouris[1] definió los sistemas distribuidos como "sistemas en los que los componentes hardware y/o software existentes en una red de computadoras, se comunican y coordinan sus acciones mediante el intercambio de mensajes".

Este concepto se ha popularizado durante los últimos años, debido a varios factores. El primer factor que impulsó el desarrollo de sistemas distribuidos fue la aparición de redes locales de alta velocidad. Otro factor importante ha sido el avance tecnológico en las prestaciones de los ordenadores personales y el desarrollo de software para soportar aplicaciones distribuidas.

Los sistemas distribuidos se ligan con el concepto de Web e Internet. Con la evolución de la Web, y la aparición de nuevas áreas, interacciones, necesidades y aplicaciones, surge el concepto de Web 2.0. Este término fue acuñado por Tim O'Reilly en 2004 para referirse a una segunda generación en la historia de la Web basada en comunidades de usuarios y una gama especial de servicios, como las redes sociales, los blogs, los wikis o las folcsonomías, que fomentan la colaboración y el intercambio ágil de información entre los usuarios.

Las primeras arquitecturas que pueden considerarse orientadas a servicio (SOA) se basaban en CORBA (*Common Object Request Broker Architecture*), que actuaba como una capa de abstracción para interconectar los distintos elementos de la arquitectura y construir los servicios. Otras tecnologías anteriores fueron DCOM (*Distributed Component Object Model*) o RPC (*Remote Procedure Protocol*). Con la necesidad de diseñar e implementar sistemas distribuidos en la Web de forma eficiente, surgen diversos desafíos, algunos centrados en el propio desarrollo (rendimiento, experiencia de usuario, etc.) y otros complementarios, basados en la reusabilidad y compatibilidad de los servicios. Sobre estas necesidades aparecen los Servicios Web, proporcionando mecanismos estándar para interconectar a los distintos usuarios con los servidores de información.

Como veremos estos servicios extienden las características básicas de las Arquitecturas Orientadas a Servicios, focalizando sus objetivos en ser completamente interoperables, además de reusables, destacando también por no ser excesivamente eficientes en relación a su rendimiento.

La distribución del presente trabajo es la siguiente:

- En el siguiente capítulo veremos qué son las arquitecturas SOA (orientadas a servicio).
- Una vez revisados sus principales conceptos, estudiaremos en el capítulo 3 los Servicios Web, y más concretamente las dos líneas actuales más destacadas: los Servicios Web SOAP y los Servicios Web REST.
- En el capítulo 4, se revisarán las principales características de los Servicios Web SOAP y REST, indicando sus principales beneficios y limitaciones. Además, se expondrá de forma introductoria el debate SOAP vs REST.
- En el capítulo 5 intentaremos detallar el trabajo existente sobre el rendimiento en los Servicios Web, aunque no sea el principal objetivo de estos sistemas, es un concepto fundamental a la hora de ofrecer cualquier servicio.
- Por último, las conclusiones serán realizadas en el capítulo 6.

[1] http://www.coulouris.net/

2. ARQUITECTURAS ORIENTADA A SERVICIOS

Según [1] SOA (Service-Oriented Architecture o Arquitectura Orientada a Servicios en castellano) son arquitecturas de software que definen el uso de servicios como soporte a los requisitos del negocio, cuyo objetivo es alcanzar el mínimo acoplamiento posible entre agentes software. Un servicio es una unidad de trabajo realizada por un proveedor de servicios para alcanzar un resultado final deseado por un consumidor del servicio. Tanto el proveedor como el consumidor del servicio son roles realizados por agentes software en lugar de sus propietarios.

En un sentido general, una arquitectura orientada a servicios es una solución software que pretende permitir a la empresa organizar y hacer uso de múltiples procesos. Con SOA, las aplicaciones software ya no son enormes bloques de funciones y procesos. En cambios, estas aplicaciones se componen de servicios modulares ensamblados. Recordemos que un servicio es una función software simple (como por ejemplo, cancelar la reproducción de un CD). Puede ser ejecutada bajo demanda por cualquier sistema, sin tener en cuenta el sistema operativo, plataforma, lenguaje de programación o posición geográfica.

Lo que es revolucionario acerca de SOA no es el concepto en sí mismo, el cual ha estado presente desde hace tiempo, sino el hecho de que se puede implementar a través de la WWW (World Wide Web). De la misma forma que las páginas web se cargan en cualquier plataforma, los servicios web trabajan de forma similar, sin tener en cuenta la plataforma, ya que se construyen utilizando estándares universales.

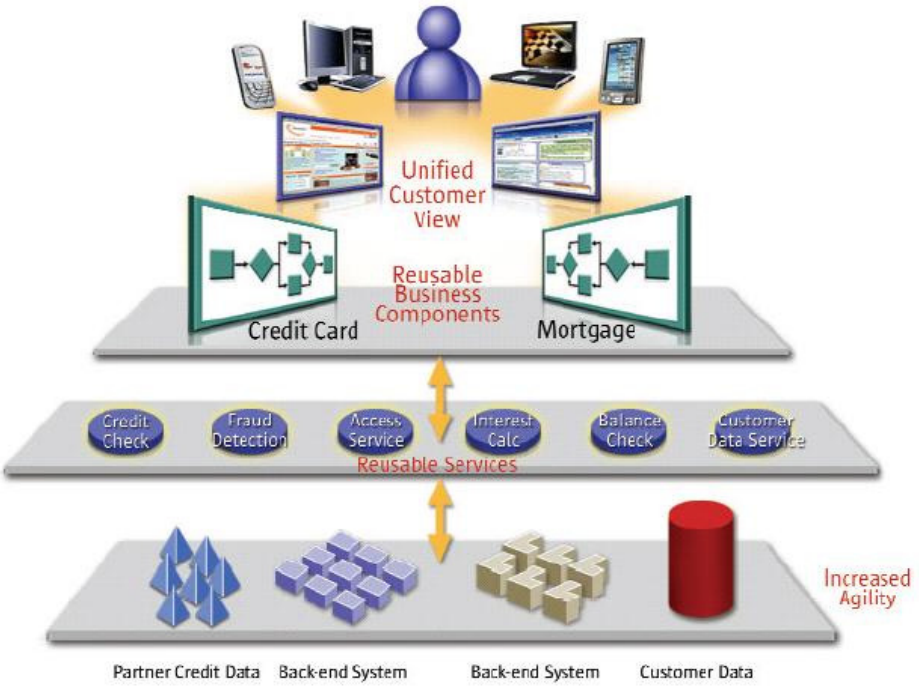

Figura 1: Arquitectura orientada a servicios [2]

Suena demasiado abstracto, pero realmente SOA se encuentran actualmente en todos los lugares. Por ejemplo, si tenemos un reproductor MP3 y queremos escuchar una canción, lo encendemos y damos al play en la selección que queramos. Así el reproductor nos ofrece el servicio de reproducción de canciones. Este reproductor puede ser reemplazado por una minicadena que

soporte reproducción de MP3 desde un dispositivo USB externo, ofreciendo el mismo servicio de reproducción de canciones, pero como mejor calidad y distintas características.

La idea de SOA parte directamente de la programación orientada a objetos, la cual sugiere una relación entre los datos y su procesamiento. Así, definen los servicios formalmente a través de interfaces independientes de la plataforma subyacente y del lenguaje de programación. Estas interfaces ocultan las particularidades de su implementación, lo que las hace independientes del desarrollador y del lenguaje de programación. A través de estas arquitecturas se consiguen diseñar e implementar sistemas altamente escalables que reflejan la lógica de negocio y a la vez facilitan la interacción entre distintos sistemas propietarios o de terceros. La razón por la cual queremos que alguien haga el trabajo por nosotros es porque es experto. Utilizar un servicio es generalmente más barato y efectivo que hacerlo por nosotros mismos. Así, la mayoría de nosotros comprendemos que no podemos ser expertos en todo. La misma regla se puede aplicar a la construcción de sistemas software.

¿Cómo consigue SOA desacoplar los agentes software que interactúan? Utilizando dos principios en la definición de la arquitectura:

1. Escoger un pequeño conjunto de interfaces simples y distribuidas para todos los agentes software participantes. Sólo la semántica genérica es codificada en las interfaces. Éstas deberían estar disponibles universalmente para todos los proveedores y consumidores.

2. Definir mensajes descriptivos por un esquema extensible que se entrega a través de las interfaces. No se ofrece información sobre el funcionamiento del sistema a través de mensajes. Los esquemas son los encargados de limitar el vocabulario y estructura de los mensajes.

Las interfaces son tremendamente importantes, si éstas no han sido bien definidas o no funcionan, el sistema no funciona. Integrar más interfaces es caro, y además incrementa la posibilidad de errores en aplicaciones distribuidas. Las interfaces remotas son la parte más lenta de la mayoría de las aplicaciones distribuidas. Con todo esto, en lugar de construir nuevas interfaces para cada aplicación, tiene más sentido reutilizar unas genéricas para todas las aplicaciones.

Así, ya que sólo disponemos de unas pocas interfaces genéricas disponibles, debemos incluir dentro de los mensajes semántica específica sobre las aplicaciones. Podemos enviar cualquier tipo de mensaje sobre nuestras interfaces, pero hay unas pocas reglas a seguir para poder decir que una arquitectura es orientada a servicio.

- Primero, los mensajes deben ser descriptivos, más que instructivos, porque el proveedor del servicio es responsable de resolver el problema. Por ejemplo, sería similar a la situación de ir a un restaurante y decir al camarero lo que te gustaría tomar y tus preferencias, pero no deberíamos explicar al cocinero cómo cocinar tus platos, paso por paso.

- Segundo, los proveedores de servicio no serán capaces de entender tu petición si tus mensajes no están escritos en un formato, estructura y vocabulario comprensible por todos los involucrados. Así, limitar el vocabulario y la estructura de los mensajes es una necesidad para lograr una comunicación eficiente. Cuanto más restringido sea el mensaje,

más sencillo es entenderlo.

- Tercero, la posibilidad de extensiones es de vital importancia. El mundo es un sitio cambiante en todo momento, y de forma similar es el entorno en el que vive el software. Estos cambios demandan cambios correspondientes en el sistema software, consumidores de servicios, proveedores, y los mensajes que intercambian. Si los mensajes no son extensibles, los consumidores y proveedores estarán bloqueados en una versión concreta del servicio. Restricción y extensibilidad están profundamente relacionadas, se necesitan ambas, e incrementar una supone una reducción de la otra. Lo ideal es lograr un correcto balance.

- Cuarto, una SOA debe poseer un mecanismo que permita al consumidor descubrir un proveedor de servicios bajo el contexto de un servicio buscado por el consumidor. El mecanismo debe ser flexible, y no debería ser un registro centralizado.

Existen además un número de posibilidades adicionales que se pueden aplicar en SOA para mejorar su escalabilidad, rendimiento y fiabilidad:

- Servicio stateless (sin estados): Cada mensaje que envía un consumidor a un proveedor debe contener toda la información necesaria para que el proveedor la procese. Esta restricción hace más escalable a un proveedor de servicio puesto que el proveedor no tiene que almacenar información de estado entre peticiones. Esto se puede denominar “servicio de producción en masa” ya que cada petición puede ser tratada de manera genérica. Esta restricción además provee de más visibilidad, ya que cualquier software de monitorización puede inspeccionar una petición independiente y extraer su intención. Esto permite además la recuperación de forma sencilla de fallos parciales, puesto que no hay estados intermedios, haciendo el servicio más fiable.

- Servicio stateful (con estados): Los servicios con estados son necesarios en diversas situaciones. Por ejemplo, al establecer una sesión entre el consumidor y el proveedor. Las sesiones se establecen típicamente por razones de eficiencia. Enviar un certificado de seguridad en cada petición es una carga importante, tanto para el consumidor como para el proveedor, es mucho más rápido reemplazar el certificado con un token compartido entre ambos. Otra situación es la de ofrecer un servicio personalizado. Estos servicios requieren que tanto el consumidor con el proveedor compartan el mismo contexto, incluido en o referenciado mediante mensajes intercambiados entre el consumidor y proveedor. El inconveniente de este requisito es que puede reducir la escalabilidad del proveedor de servicio, ya que puede necesitar recordar el contexto para cada consumidor. Además, incrementa el acoplamiento entre el proveedor de servicio y el consumidor, y hace que el cambio entre proveedores sea más difícil.

- Peticiones idempotentes (que no realizan ningún cambio): Las peticiones duplicadas recibidas por un agente software tienen el mismo efecto que una petición única. Este requisito permite que los proveedores y consumidores incrementen la fiabilidad total del servicio, simplemente repitiendo la petición si ha habido algún fallo.

Podemos decir entonces, que SOA no es una herramienta, sino un conjunto de patrones de construcción de nuevas aplicaciones más dinámicas y menos dependientes. SOA facilita el acceso a la lógica de negocios y la información entre diversos servicios de una manera sistemática,

pudiendo además orquestar diversos servicios remotos para componer uno único. La mayoría de las definiciones identifican la utilización de Servicios Web, los cuales veremos en el siguiente capítulo, en la implementación de una arquitectura orientada a servicios. No obstante, se puede implementar utilizando cualquier otra tecnología basada en servicios.

3. SERVICIOS WEB

Los Servicios Web se han convertido en la implementación más utilizada en arquitecturas orientadas a servicios. Esto se debe a que poseen un conjunto de características que permiten cubrir todos los principios básicos de la orientación a servicios.

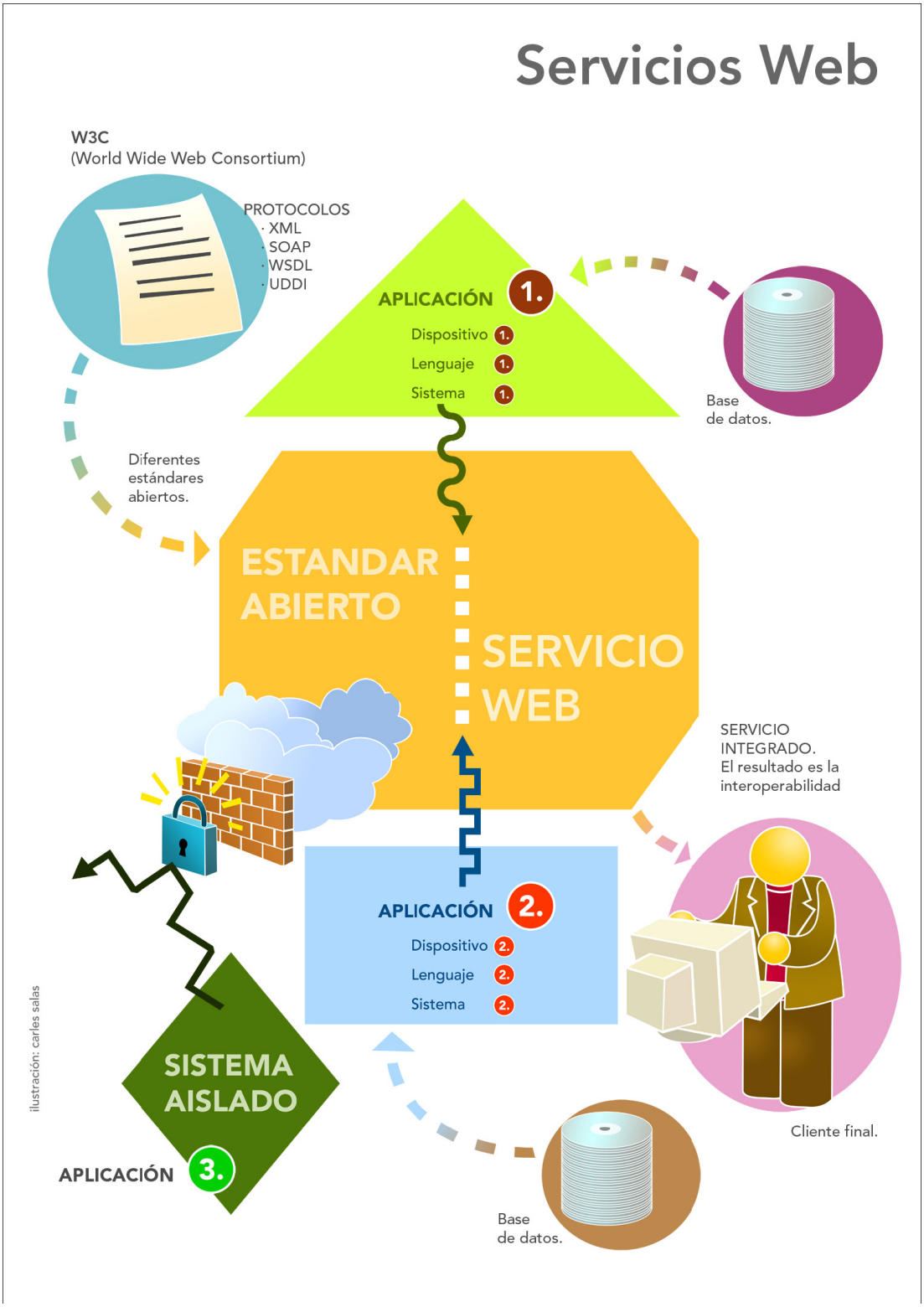

Figura 2: Los Servicios Web se basan en estándares abiertos para interconectar aplicaciones y usuarios independientemente de las plataformas [3]

El concepto de Servicio Web puede resultar confuso, ya que no existe una definición única universalmente aceptada sobre lo que son y lo que el concepto engloba. Los Servicios Web surgieron como un conjunto de protocolos, estándares y recomendaciones, definidos por la W3C [5] *(World Wide Web Consistorium*) y OASIS [6] (*Organization for the Advancement of Structured Information Standards*), para lograr la interoperabilidad en la interacción entre máquinas, sistemas software y aplicaciones a través de la red. La definición de Servicio Web ha estado siempre bajo debate en el grupo de trabajo de la arquitectura de Servicios Web del W3C [7].

Aunque no esté totalmente cerrado, se acepta generalmente que un Servicio Web es una SOA con restricciones adicionales:

1. Las interfaces se deben basar en protocolos de Internet como HTTP, FTP y SMTP.

2. Los mensajes deben ser XML, excepto para datos anexos binarios.

Así, una posible definición sería hablar de ellos como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web.

Estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.

Aunque podemos encontrar diversos estilos de Servicios Web, en este estudio nos centraremos en aquellos basados en SOAP (o denominados comúnmente WS), y en REST, descartando por ejemplo otras líneas como XML-RPC [8] (considerado el precursor de SOAP).

3.1. SOAP WS

Un Servicio Web SOAP introduce las siguientes restricciones sobre las características ya citadas de SOA:

1. Excepto para datos binarios anexos, los mensajes deben ser transportados sobre SOAP.

2. La descripción de un servicio debe ser hecha en WSDL.

3. Uso de UDDI, que son las siglas del catálogo de negocios de Internet denominado *Universal Description, Discovery and Integration*.

El siguiente gráfico muestra cómo interactúa un conjunto de Servicios Web:

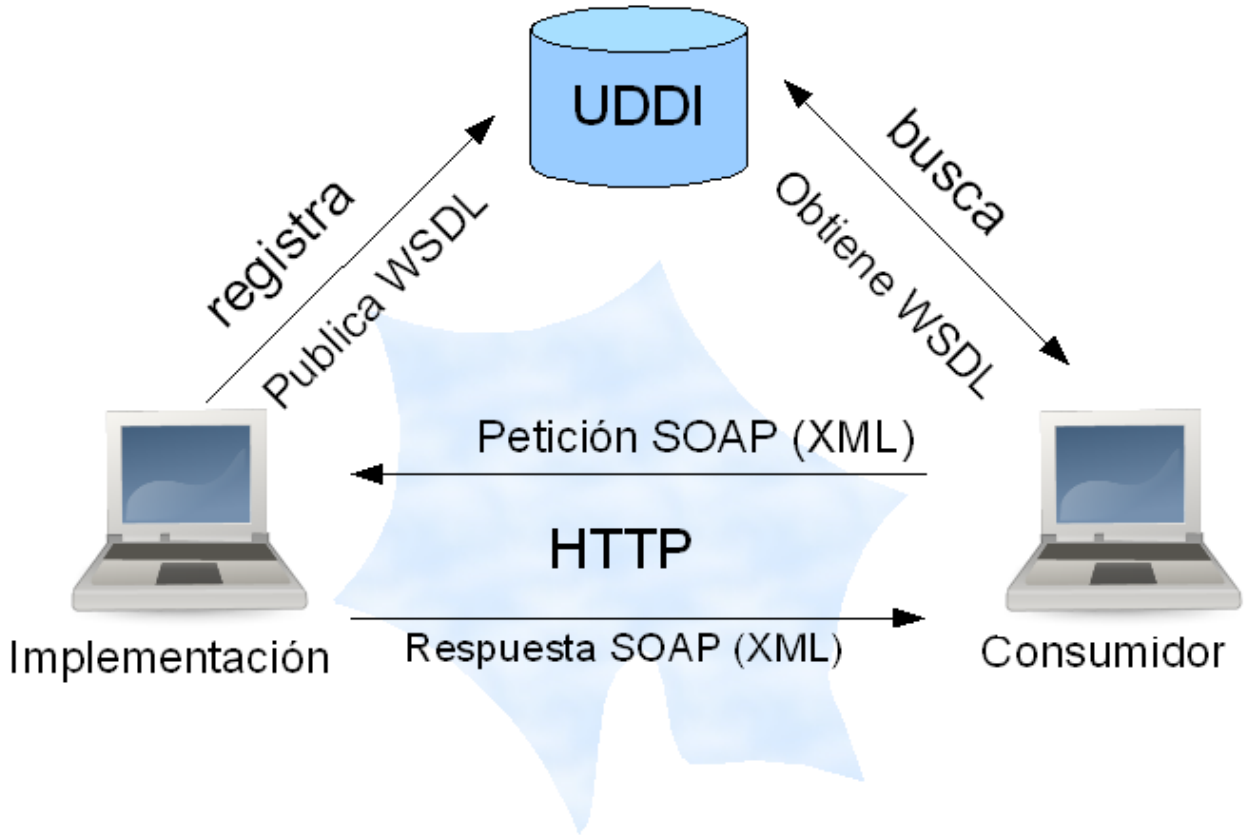

Figura 3: Interacción a través de Servicios Web [9]

Vemos que en todo este proceso intervienen una serie de tecnologías que hacen posible esta circulación de información. Un Servicio Web está formado por los siguientes componentes:

- Lógica. Se trata del componente que procesa la petición para generar la información solicitada por el cliente. Básicamente resuelve el "problema" y puede, para ello, comunicarse con otros Servicios Web, acceder a bases de datos o bien invocar API de otras aplicaciones solicitando la información (o parte de ella) que ha de generar para enviar en formato XML.

- SOAP (Simple Object Access Protocol). Protocolo de comunicación, basado en XML, que sirve para la invocación de los Servicios Web a través de un protocolo de transporte, como HTTP (ó SMTP, etc.). Consta de tres partes: una descripción del contenido del mensaje, unas reglas para la codificación de los tipos de datos en XML y una representación de las llamadas RPC para la invocación y respuestas generadas por el Servicio Web. El mensaje SOAP está compuesto por un *envelope* (sobre), cuya estructura está formada por los siguientes elementos: *header* (cabecera) y *body* (cuerpo).

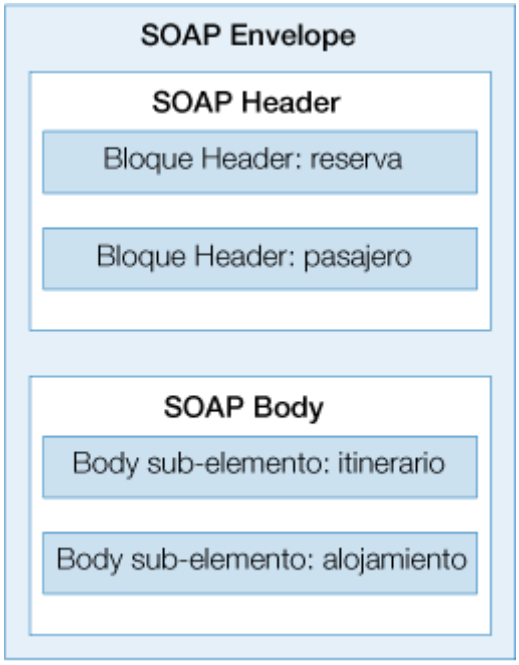

Figura 4: Estructura de los mensajes [10]

- UDDI (Universal Description, Discovery and Integration): Directorio donde es posible publicar los Servicios Web, permitiendo con ello que los posibles usuarios de ese servicio puedan obtener toda la información necesaria para la invocación y ejecución del Servicio Web. Un directorio UDDI ofrece una serie de interfaces que posibilitan tanto la publicación como la obtención de información sobre los Servicios Web publicados. La información registrada se clasifica según lo que se desee obtener del servicio:
  - Información de negocio: acerca de quién publica el servicio.
  - Información de servicio: descripción del tipo de servicio.
  - Información de enlace: dirección (URL, por ejemplo) para acceder al servicio.

- WSDL (Web Services Description Language). Lenguaje basado en XML que permite la descripción de los Servicios Web definiendo la gramática que se debe usar para permitir su descripción y capacidades (datos, comandos que aceptan o producen), y su publicación en un directorio UDDI.

Veamos ahora un ejemplo concreto, donde una serie de Servicios Web interactúan para ofrecer una aplicación común:

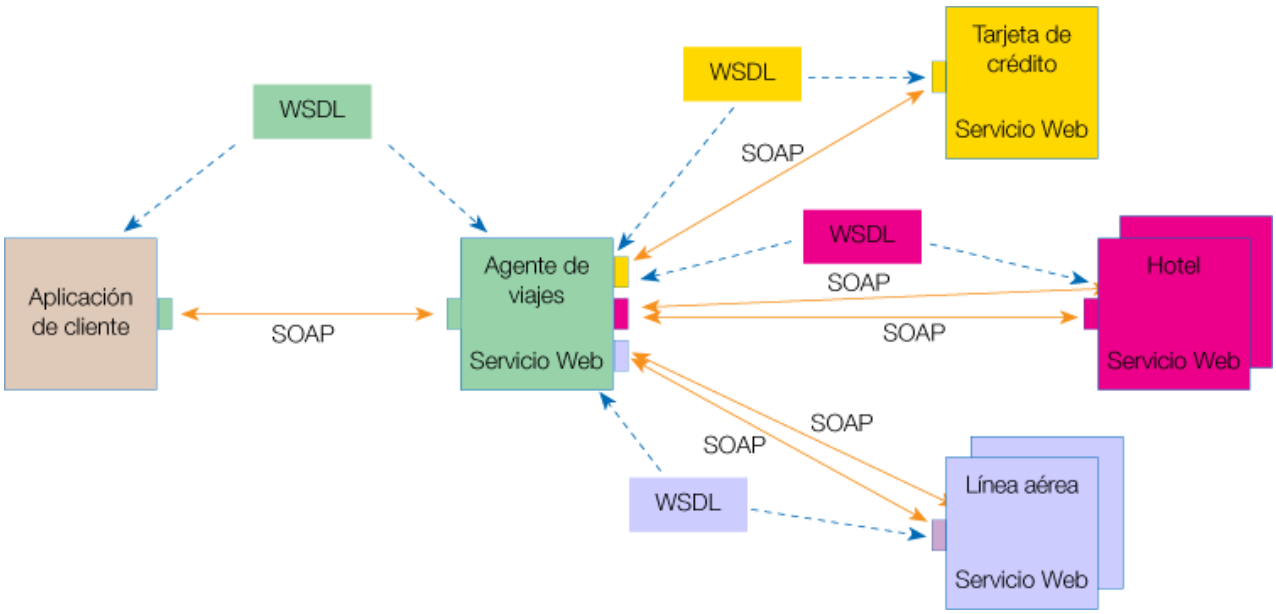

Figura 5: Los servicios Web en Funcionamiento [11]

Según el ejemplo de la imagen anterior, un usuario solicita, a través de una aplicación, información sobre un viaje que desea realizar haciendo una petición a una agencia de viajes que ofrece sus servicios a través de Internet. La agencia de viajes ofrecerá a su cliente (usuario) la información requerida. Para proporcionar al cliente la información que necesita, esta agencia de viajes solicita a su vez información a otros recursos (otros Servicios Web) en relación con el hotel y la compañía aérea. La agencia de viajes obtendrá información de estos recursos, lo que la convierte a su vez en cliente de esos otros Servicios Web que le van a proporcionar la información solicitada sobre el hotel y la línea aérea. Por último, el usuario realizará el pago del viaje a través de la agencia de viajes que servirá de intermediario entre el usuario y el Servicio Web que gestionará el pago.

A continuación se muestra el código que se utilizaría para solicitar un viaje:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
 <m:reserva xmlns:m="http://empresaviajes.ejemplo.org/reserva"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
   <m:referencia>
      uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d
   </m:referencia>
   <m:fechaYHora>2001-11-29T13:20:00.000-05:00</m:fechaYHora>
  </m:reserva>
  <n:pasajero xmlns:n="http://miempresa.ejemplo.com/empleados"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
   <n:nombre>Pepe Ejemplo</n:nombre>
  </n:pasajero>
 </env:Header>
<env:Body>
  <p:itinerario
    xmlns:p="http://empresaviajes.ejemplo.org/reserva/viaje">
   <p:ida>
     <p:salida>Nueva York</p:salida>
     <p:llegada>Los Angeles</p:llegada>
     <p:fechaSalida>2001-12-14</p:fechasalida>
     <p:horaSalida>última hora de la tarde</p:horaSalida>
```

```
      <p:preferenciaAsiento>pasillo</p:preferenciaAsiento>
    </p:ida>
    <p:vuelta>
      <p:salida>Los Angeles</p:salida>
      <p:llegada>Nueva York</p:llegada>
      <p:fechaSalida>2001-12-20</p:fechaSalida>
      <p:horaSalida>media-mañana</p:horaSalida>
      <p:preferenciaAsiento/>
    </p:vuelta>
  </p:itinerario>
  <q:alojamiento
   xmlns:q="http://empresaviajes.example.org/reserva/hoteles">
   <q:preferencia>ninguna</q:preferencia>
  </q:alojamiento>
 </env:Body>
</env:Envelope>
```

Durante la evolución de las necesidades de las aplicaciones basadas en Servicios Web, por ejemplo de las grandes organizaciones, se han desarrollado mecanismos que permiten enriquecer las descripciones de las operaciones que realizan sus servicios mediante anotaciones semánticas y con directivas que definen el comportamiento. Son las denominadas extensiones, que veremos de forma resumida en el siguiente apartado.

3.1.1. WS-*

La simplicidad si bien es el punto fuerte de los servicios Web, también en determinados casos puede ser su talón de Aquiles, ya que operaciones como asegurar las comunicaciones o enviar archivos binarios pueden convertirse en tareas muy complejas.

SOAP puede ser extendido realizando adiciones de módulos de funcionalidad. Este enfoque permite a los desarrolladores usar los módulos y funcionalidad que ellos necesitan, sin tener la necesidad de implementar la totalidad de estos.

Algunas de las extensiones, denominadas Ws-*, que pueden ser deseables en los proveedores son las siguientes:

- Attachments – Permite incluir documentos no XML, como archivos binarios o imágenes.

- Routing/Intermediaries – Relacionadas al proceso de enviar mensajes SOAP a través de intermediarios. Permite agregar varios Servicios Web (WS) y ofrecerlos como parte del paquete, incrementando la escalabilidad de los servicios.

- Security – Da un marco de seguridad a la comunicación. Así el SOAP envelope soporta integridad del mensaje, confidencialidad y autenticación. Describe certificados X.509, tickets Kerberos, etc.

- Quality of Services – QoS es una medida que mide la calidad del servicio, con esta extensión es posible caracterizar el funcionamiento del servicio.

- Context/Privacy – Relacionada con la WS-Security, hace referencia a guardar el contexto y privacidad, del entorno de los usuarios.

Además existen otros foros encargados de ofrecer extensiones para cubrir escenarios más complejos, o incrementar la interoperabilidad entre los distintos servicios. Los trabajos más importantes que actualmente se están llevando a cabo son:

- Interoperability [12]: Este grupo se encarga de promover la interoperabilidad de los Servicios Web entre cada una de las plataformas, aplicaciones y lenguajes de programación. Se trata de un foro abierto donde se ofrecen a los clientes o posibles usuarios de guías de implementación prácticas recomendadas y soporte para el desarrollo de servicios web interoperables.

- Security [13]: trata de definir unos requisitos a aplicar en los mensajes SOAP para aumentar la seguridad en el uso de los Servicios Web garantizando la integridad de los mensajes, la confidencialidad y la autenticación.

- BPEL4WS (Business Process Execution Language for Web Services) [14]: Lenguaje para describir procesos de negocio sobre Servicios Web. Este lenguaje, combinado con las especificaciones WS-Transaction [15] y WS-Coordination [16], determina como se deben de definir, coordinar e integrar los procesos de negocio dentro de la empresa o con otros proveedores, a través de sistemas heterogéneos.

En el siguiente enlace de INOOQ [17] se puede encontrar un mapa muy completo con una revisión de los estándares relacionados con los Servicios Web.

3.2. Servicios Web REST

El término REST [18], acrónimo de REpresentational State Transfer, fue introducido por primera vez por Roy Fielding [19] (uno de los creadores de HTTP) en la lectura de su tesis para describir un tipo de arquitectura de los sistemas en red. Un Servicio Web REST es un SOA basado en el concepto de recurso. Un recurso es cualquier cosa que tiene una URI (Uniform Resource Indentifier), pudiendo tener cero o más representaciones.

Un Servicio Web REST tiene las siguientes características:

1. Las interfaces deben construirse sobre HTTP. Las siguientes funciones son definidas:
   - HTTP GET: Usado para obtener una representación de un recurso. Un consumidor lo utiliza para obtener una representación desde una URI. Los servicios ofrecidos a través de este interfaz no deben contraer ninguna obligación respecto a los consumidores.
   - HTTP DELETE: Se usa para eliminar representaciones de un recurso.
   - HTTP POST: Usado para actualizar o crear las representaciones de un recurso.
   - HTTP PUT: Se usa para crear representaciones de un recurso.
2. La mayoría de los mensajes son XML, definidos por un esquema XML.
3. Mensajes simples se pueden codificar en las URL.
4. Los servicios y los proveedores de servicios deben ser recursos, mientras que los consumidores pueden ser un recurso.

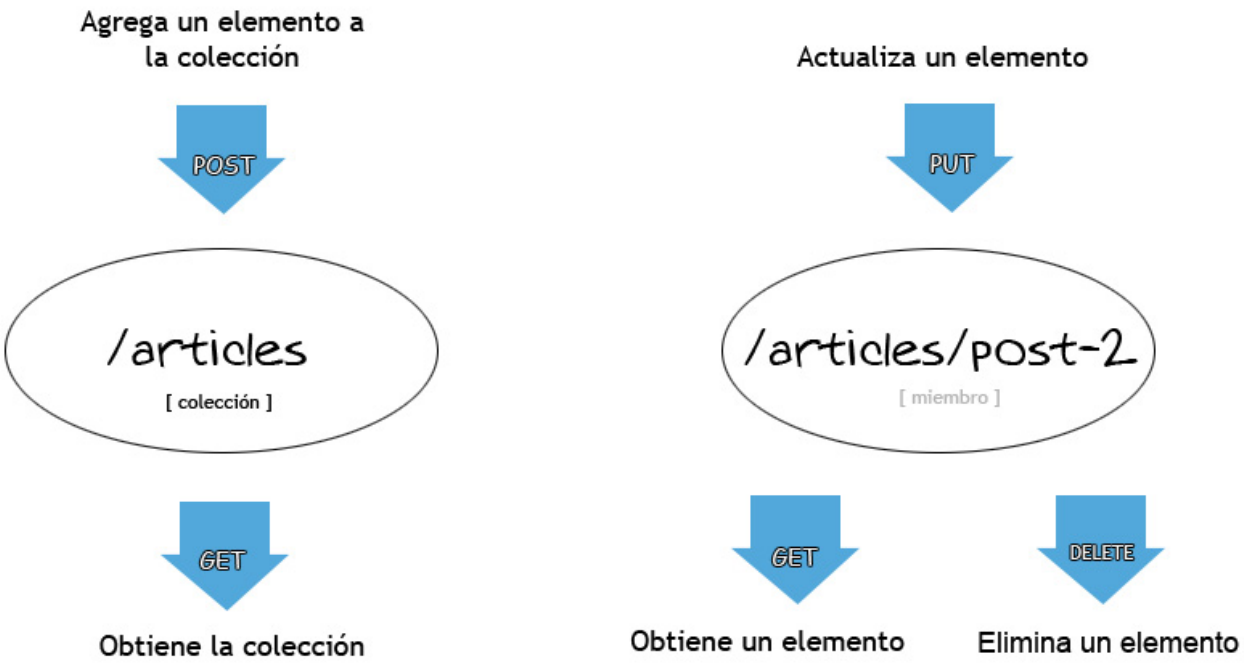

Figura 6: Ejemplo con las funciones de REST [20]

Los Servicios Web REST requieren poca infraestructura, aparte de las tecnologías HTTP estándar y XML, que actualmente son soportadas por la mayoría de los lenguajes de programación y plataformas. Los servicios web REST son simples y efectivos, ya que HTTP es el interfaz más extendido y es soportado por la mayoría de las aplicaciones.

3.2.1. Arquitectura REST

La Web por tanto, se compone de recursos, y como hemos dicho un recurso es cualquier objeto de interés, identificado por una o varias URIs. Por ejemplo, la compañía de venta online AlbertoL S.A. puede definir un recurso "ordenador", y los clientes acceder a ese recurso mediante la siguiente URL:

http://www.albertolsa.com/productos/ordenador

La representación del recurso sería devuelta al cliente (por ejemplo, productosOrdenador.html). Esta representación sitúa al cliente en un estado. Cuando se accede a la representación del recurso, la aplicación del cliente realiza una transferencia de estado (*Representational State Transfer*).

La motivación para REST fue capturar las características de la Web que hacían de la propia Web un éxito. Estas características están siendo ahora usadas para guiar la evolución de la Web.

REST no es un estándar, sino que es un estilo de arquitectura (de forma análoga, no hay un estándar cliente-servidor, sino que hay un tipo de arquitectura cliente-servidor). Sin ser un estándar, REST hace uso de estándares:

- HTTP [RFC 1945]: HyperText Transfer Protocol.
- URL [RFC 1738] (*Uniform Resource Locator*): Mecanismo de identificación de recursos.

- XML / HTML / PNG / etc.: Distintos formatos de representación de recursos.
- Tipos MIME: Como text/xml, text/html, image/png, etc.

Algunas características de REST son:

- Cliente – Servidor: Estilo de interacción basada en pull (bajo demanda).
- Sin estado: Cada petición desde un cliente hacia un servidor debe contener toda la información necesaria para comprender la petición, y no puede tomar provecho de ningún contexto almacenado en el servidor.
- Caché: Para mejorar la eficiencia de la red, las respuestas deben ser capaces de ser etiquetadas como cacheables o no cacheables.
- Interfaz uniforme: Todos los recursos son accesibles mediante un interfaz genérico (por ejemplo, HTTP GET, POST, PUT, DELETE).
- Recursos con nombres: El sistema se compone de recursos que son nombrados usando una URL.
- Representaciones de recursos interconectados: Las representaciones de los recursos están interconectadas usando URLs, por tanto un cliente está capacitado para progresar de un estado a otro.
- Componentes en capas: Intermediarios, tales como servidores proxy, servidores caché, gateways, etc., pueden ser introducidos entre los clientes y los recursos, para ofrecer rendimiento, seguridad, etc.

3.2.2. Ejemplo

Imaginemos que la empresa anteriormente citada, AlbertoL S.A. (empresa de componentes informáticos), ha desplegado varios servicios para permitir que sus clientes puedan:

- Obtener una lista de componentes.
- Obtener información detallada sobre un componente en particular.
- Enviar una orden de compra.

Veamos cómo estos servicios se implementarían sobre REST.

Obtener una lista de componentes

El Servicio Web dispone de una URL para un recurso con la lista de componentes. Por ejemplo, el cliente usaría la siguiente URL para obtenerla:

```
http://www.albertolsa.com/componentes
```

Notar que “cómo” el Servicio Web genera la lista de componentes es completamente transparente al cliente. Todo lo que el cliente sabe es que ha solicitado la URL anterior y que el documento que contiene la lista de componentes le ha sido devuelto. Puesto que la implementación es transparente a los clientes, AlbertoL S.A. es libre de modificar la implementación que hay por debajo de este recurso, sin influenciar a los clientes. Esto es lo que denominamos bajo acoplamiento.

Éste es un ejemplo del documento que el cliente recibe:

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.albertolsa.com"
         xmlns:xlink="http://www.w3.org/1999/xlink">
      <Part id="00345"
xlink:href="http://www.albertolsa.com/componente/00345"/>
      <Part id="00346"
xlink:href="http://www.albertolsa.com/componente/00346"/>
      <Part id="00347"
xlink:href="http://www.albertolsa.com/componente/00347"/>
      <Part id="00348"
xlink:href="http://www.albertolsa.com/componente/00348"/>
</p:Parts>
```

Asumimos que el servicio ha determinado (a través de procesos de negociación) que el cliente quiere la representación como XML (para procesamiento M2M). Vemos que la lista de componentes tiene enlaces para obtener información detallada sobre cada uno. Esto es una característica clave de REST, el cliente se traslada de un estado al siguiente examinando y eligiendo las URLs de entre las alternativas que dispone el documento respuesta.

Obtener la información detallada de un componente

El Servicio Web ofrece una URL para cada recurso componente. Por ejemplo, veamos una petición del componente 00345:

```
http://www.albertolsa.com/componente/00345
```

Este sería el documento que recibiría el cliente:

```
<?xml version="1.0"?>
<p:Part xmlns:p="http://www.albertolsa.com"
        xmlns:xlink="http://www.w3.org/1999/xlink">
      <ID>00345</ID>
      <Nombre>Teclado</Nombre>
      <Descripcion>Teclado marca Manzana(R)</Descripcion>
      <Especificacion
xlink:href="http://www.albertolsa.com/componente/00345/specif
ication"/>
      <CosteUnidad moneda="EUR">10.00</CosteUnidad>
      <Cantidad>10</Cantidad>
</p:Part>
```

De nuevo se observa como esta información está enlazada aún a más datos (la especificación para este componente se puede encontrar yendo al enlace). Cada documento respuesta permite al cliente ir a una información más detallada.

Enviar una orden de compra

El Servicio Web tiene disponible una URL para enviar una orden de compra. El cliente debe crear un documento de instancia de orden de compra, que debe ajustarse al esquema que la empresa ha diseñado para este proceso. El cliente envía OC.xml mediante un HTTP POST.

El servicio de orden de compra, responde al HTTP POST con una URL, donde el cliente puede encontrar la información de la orden en cualquier momento y así editarla o actualizarla. La

orden de compra pasa a ser un fragmento de información compartido entre el cliente y el servido como un Servicio Web.

4. CARACTERÍSTICAS SOAP y REST

Como ya se ha ido comentando durante el trabajo, los Servicios Web ofrecen varios beneficios sobre otros tipos de arquitecturas de computación distribuidas [21]:

- Interoperabilidad: Ésta es la ventaja más importante de los Servicios Web. Éstos típicamente trabajan fuera de redes privadas, ofreciendo a los desarrolladores rutas no propietarias a sus soluciones. Por tanto, los servicios suelen tener mayor vida útil, recibiendo mayor retorno ante la inversión en el servicio desarrollado. Además, los desarrolladores pueden optar por su lenguaje de programación favorito, puesto que su implementación está soportada sobre la mayoría de las tecnologías existentes. Por último, gracias al uso de métodos estándar de comunicaciones, los Servicios Web son virtualmente independientes de la plataforma.

- Usabilidad: Los servicios Web permiten que la lógica de negocio de diferentes sistemas puedan ser expuestas sobre la Web. Esto ofrece a las aplicaciones la libertad de elegir el Servicio Web que necesitan. En vez de reinventar la rueda para cada cliente, sólo es necesario incluir la lógica de negocio específica a la aplicación en el lado del cliente. Esto permite que se desarrolle tanto el servicio como el código del lado del cliente usando los lenguajes y herramientas que se deseen.

- Reusabilidad: Los Servicios Web no ofrecen directamente un modelo de desarrollo de aplicaciones, pero casi, ya que se dispone de una aproximación que necesita un mínimo desarrollo de código para el desarrollo de dichos servicios. Esto facilita la reutilización de componentes en otros servicios.

- Despliegue: Los Servicios Web se despliegan sobre tecnologías de Internet estándar. Esto hace posible desplegar Servicios Web sobre firewalls hasta sobre servidores corriendo en Internet en la otra parte del globo. Además, gracias al uso de estándares, la seguridad es posible (mediante el uso de SSL por ejemplo).

Pero podemos también encontrar inconvenientes o limitaciones:

- Aunque la simplicidad de los Servicios Web es una ventaja, en algunos aspectos puede ser un estorbo. Los Servicios Web usan protocolos basados en texto plano que usan un método demasiado pesado para identificar la información. Esto significa que en ocasiones las peticiones son más largas que las peticiones codificadas con un protocolo binario. El tamaño extra es ciertamente sólo una cuestión a tratar sobre conexiones lentas, o conexiones extremadamente saturadas, pero es algo a considerar.

- Tanto HTTP como HTTPs (el núcleo de los protocolos Web) son simples, pero no fueron inicialmente considerados para sesiones largas. Típicamente, un navegador realiza una conexión HTTP, solicita una página web y después se desconecta. En entornos CORBA o RMI, un cliente se conecta al servidor, pudiendo permanecer conectado durante un periodo extenso de tiempo, recibiendo información periódica en el cliente. Esta interacción es difícil de obtener con los Servicios Web, y es necesario un trabajo extra para conseguirlo.

- El problema con HTTP y HTTPS cuando se emplean para sustentar Servicios Web es que estos protocolos son stateless, sin estado, la interacción entre el servidor y el cliente es típicamente breve, y cuando no hay datos siendo intercambiados, el servidor y el cliente no tienen conocimiento sobre el otro. Más específicamente, si el cliente hace una petición al servidor, recibirá información, y si inmediatamente cae debido a un corte de corriente, el servidor nunca sabrá que el cliente ya no sigue activo. El servidor necesita una forma de mantener un seguimiento sobre lo que el cliente está realizando y también determinar cuándo un cliente ya no está activo.

- Típicamente, un servidor envía algún tipo de identificador de sesión al cliente cuando éste accede por primera vez al servicio. El cliente usa este identificador cuando realiza peticiones al servidor. Esto permite al servidor recuperar cualquier tipo de información que tenga sobre el cliente. Además, el servidor debería utilizar un mecanismo de timeout para determinar cuándo un cliente no está activo. Si el servidor no recibe una petición desde un cliente pasado un determinado tiempo, se asume que el cliente está inactivo y se elimina la información que se estaba manteniendo. Este overhead extra, significa mayor trabajo para los desarrolladores de los Servicios Web.

- Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).

- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA, o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.

Este último hecho se acentúa en los Servicios Web sobre SOAP, ya que las cabeceras del mensaje introducen un overhead considerable [22], cosa que no ocurre con REST. Veamos en la siguiente sección un resumen del debate existente en la actualidad sobre ambas filosofías.

4.1. Debate SOAP vs REST

El debate sobre SOAP y REST está en auge [24][25]. En definitiva, son soluciones para un mismo problema, la integración de sistemas. Empresas como Google [26], Facebook y otros han tomado ya la opción REST para la implementación de sus Servicios Web.

Los RESTful Web Services han demostrado su capacidad para responder de manera efectiva a los requerimientos de publicación y sindicación de contenido y medios. Sin embargo, SOAP encaja mejor en soluciones con un mayor alcance y requisitos mayores, tanto en número de servicios/operaciones, número de aplicaciones cliente, número de equipos de desarrollo implicados y complejidad de mensajes, principalmente enfocados al desarrollo empresarial.

Si bien en Internet las grandes empresas están tendiendo a ofrecer sus servicios sobre REST, debido principalmente a la sencillez de invocación que ofrecen, y su limitado overhead (en los servicios sobre SOAP, las cabeceras y datos que no son la información relevante a transmitir

ocupan gran parte del ancho de banda), podemos decir que no siempre REST es la mejor opción. Por ejemplo, REST lleva asociado HTTP como único protocolo de transporte. Si tenemos necesidad de utilizar otro tipo de transporte como JMS, SOAP sí es válido, pero no así el caso de REST. Además, en niveles de seguridad, la pila WS-* (concretamente con su definición de WS-Security a nivel de mensaje) permite un mayor nivel de seguridad que REST (que se implementa en caso de necesidad a nivel de HTTP como transporte).

Las extensiones WS-* son una de las características que adolecen los servicios REST. Pero cada vez existen más iniciativas para poder cubrir estas posibles "lagunas". En relación a la seguridad, Amazon apuesta por el estándar HMAC [28], para poder autenticar las peticiones REST, por lo que quizás en el futuro veamos que REST puede suplir a los Servicios SOAP en todos los aspectos.

Mientras tanto, ambas iniciativas tienen su hueco y dependiendo de la necesidad será más recomendado utilizar unos u otros. Lo que sí es claro, es que REST va a dar mucho que hablar de aquí a poco tiempo, de hecho se está utilizando ya ampliamente (la sindicación de blogs por RSS o ATOM utilizan REST; Google expone sus servicios web como REST, eBay y Amazon ofrece una interfaz REST para sus desarrolladores, etc.), y esto supone que cuanto más se puja por una tecnología, ésta tiene la posibilidad de evolucionar más rápidamente.

Figura 7: Google Apps [29]

Una vez revisados los principales puntos de discusión y el debate SOAP vs REST, en el siguiente capítulo nos centraremos en el rendimiento, intentando estudiar la problemática y los esfuerzos realizados para mejorar este comportamiento.

5. RENDIMIENTO

Mientras SOA ofrece diversos beneficios, muchas de sus implementaciones están diseñadas de modo que no favorecen el rendimiento. SOA está diseñada sobre los principios de flexibilidad, extensibilidad y reusabilidad. Los sistemas del pasado estaban fuertemente acoplados, siendo generalmente bastante eficientes, pero frágiles y difíciles de cambiar. Tendían a no ser modulares, por lo que su mantenimiento era caro ante posibles adaptaciones.

Flexibilidad y reuso [30] son objetivos importantes en el diseño e implementación de los servicios, que permiten obtener beneficios cuantificables. La flexibilidad del sistema permite agilidad en el negocio, que puede ser la diferencia entre el éxito o el fracaso. La extensibilidad permite una integración más rápida de nuevas funcionalidades, para soportar cambios en los requisitos del negocio. El reuso puede incrementar la productividad y reducir el mantenimiento después.

Mientras que SOA ofrece estos beneficios, el diseño de muchas implementaciones no favorece el rendimiento. Rendimiento es un término que se usa a menudo para distintos conceptos:

- Scale-up (escalabilidad): La tasa de transferencia es un concepto importante que puede ser definido como el número de mensajes de un tamaño dado que pueden ser procesados en un periodo de tiempo dado. La latencia es un concepto relacionado, describe cuánto tarda un mensaje dado en viajar a través del mismo sistema. Estos términos pueden ser usados para describir la capacidad de un sistema para “scale up”, es decir, para escalarlo.

- Scale-out (distribución): Sistemas distribuidos y computación paralela son áreas estudiadas desde hace más de una década. Hay ganancias significantes cuando las cargas de trabajo se dividen en componentes que son llevadas a cabo en múltiples sitios en el mismo momento. Sin embargo, hay retos que deben ser resueltos para satisfacer las reglas requeridas por los negocios cuando el procesamiento paralelo es empleado.

Para diseñar un Servicio Web con mayor rendimiento, hay varios principios importantes que deben ser considerados:

- Utilizar la herramienta correcta para el trabajo: SOA no significa siempre Servicios Web, los cuales al hacer uso de XML y SOAP, no son tan eficientes como desarrollos propietarios. Por lo que si en un caso dado, el objetivo principal es la rapidez en la comunicación, quizás sea mejor adoptar una alternativa a los Servicios Web.

- Intentar mantener el servicio ligero: El parseo XML puede introducir un overhead importante, generando cuellos de botellas en caso de transmitir mensajes grandes entre componentes. Además, hay que examinar concienzudamente el tamaño de los mensajes XML y su posible reducción.

- Hacer más de una cosa a la vez: En el diseño de los servicios podemos considerar descomponer el procesamiento en bloques más pequeños, que trabajen de forma coordinada para incrementar el rendimiento.

- Diseñar la interfaz del Servicio Web para minimizar el tráfico de la red: Una API bien diseñada puede hacer que el cliente minimice el número de peticiones para conseguir la

información.

- Mensajes SOAP largos y complejos pueden suponer un cuello de botella, debido al tiempo que lleva parsearlos y serializarlos/deserializarlos.

- Elementos SOAP intermedios (gateways, proxys) pueden minimizar el parseo de mensajes.

- Introducir seguridad incrementa los costes de rendimiento: No todo el tráfico SOAP necesita ser seguro, y en ocasiones el coste en rendimiento de un método de seguridad end-to-end (por ejemplo, WS-Security) es mayor que el que introduce un mecanismo de seguridad a nivel de transporte (como SSL).

- La codificación binaria de cierta información en ocasiones supone un incremento de rendimiento.

- El cacheo es una forma de mejorar el rendimiento en servicios de procesado intensivo, aunque en general es aplicable sólo para servicios de sólo lectura.

- Conexiones persitentes son buenas en caso de disponer de un gran número de mensajes de tamaño reducido. Para mensajes mayores, tiene menor efectividad.

- El estilo Document/Literal de los mensajes SOAP son menores en tamaño y menos complejos que los mensajes de tipo RPC/SOAP.

De forma más concreta, cada vez es necesario disponer de aplicaciones más completas en Internet, las cuales necesitan hacer muchas llamadas a servicios empresariales, para poder presentar los resultados a los usuarios. Si los Servicios Web tardan demasiado tiempo, el *User Interfacing* (presentación de la información al usuario) llevará demasiado tiempo y por tanto se percibirá un rendimiento pobre.

Por tanto, según la aplicación o servicio, podemos eliminar todas las características no necesarias, para obtener un rendimiento mayor, a costa de reducir opciones extra, como es obvio.

De nuevo en el siguiente estudio [32], se estudia la calidad de servicio (QoS) para Servicios Web, presentando alguna de las opciones citadas en este capítulo. Básicamente y de forma resumida: Usar parsers eficientes y ligeros, mensajes XML reducidos en la medida de lo posibles, uso de tipos de datos simples en los mensajes SOAP además de usar el "cacheo" en Servicios Web, es decir, almacenar información intermedia para ser servida sin volver a procesar toda la información.

¿Cómo funcionaría este cacheo de información?

Para los Servicios Web sobre HTTP (Servicios Web XML y sobre Internet), se puede cachear los resultados del procesado para usarlos en futuras peticiones (en code google hay un espacio reservado para un proyecto en esta línea [36]). Esta posibilidad es válida principalmente para servicios de sólo lectura, como ya se ha indicado antes, que por ejemplo hacen peticiones en bases de datos que dan acceso a contenido semi-estático (catálogos de productos, información de marketing, etc.). Varias partes del mensaje SOAP se pueden usar para identificar unívocamente la petición para cachearla.

La capacidad de caching es provista definiendo políticas caching y construyendo identificadores de caché basados en:

- Acción SOAP
- SOAPEnvelope
- Componente puerto
- Operación SOAP
- Parámetros de operación SOAP

Las políticas incluyen reglas e intervalos de expiración para asegurar la validad de la información salvada. Los componentes que no necesitan el parseo del SOAP Envelope, ofrecerán la mayor ganancia de rendimiento. Por ejemplo, en la especificación SOAP se define la cabecera HTTP SOAPAction en la petición, que es usada por los servidores proxy HTTP para distribuir las peticiones a los distintos servidores HTTP. Esta cabecera puede usarse en el caso de caché para construir los IDs sin tener que parsear el mensaje SOAP. El identificador ID caché, puede entonces usarse para encontrar la respuesta del servicio de información desde la caché, para optimizar así el rendimiento.

6. CONCLUSIONES

Hemos visto que los Servicios Web surgieron ante la necesidad de disponer de un conjunto de soluciones para interconectar los servicios y usuarios en la Web. Desde su aparición, basados en las Arquitecturas Orientadas a Servicio (SOA), han ido evolucionando en pos de cubrir las necesidades que han ido apareciendo a raíz de las nuevas aplicaciones, negocios, características de los usuarios y posibilidades de la Red.

Actualmente existen dos ramas importantes en relación a los Servicios Web. Cada una dispone de una arquitectura propia, y por consiguiente su funcionamiento y características que ofrecen son distintas. Los Servicios Web basados en SOAP, se podrían catalogar como clásicos, siendo los componentes principales empleados XML (para codificar la información), SOAP (para formar los mensajes), WSDL (para definir los servicios ofrecidos, a modo de contrato entre el servidor y el cliente) y UDDI (que es un registro donde se publican los servicios). Por el contrario, los Servicios Web REST, directamente trabajan sobre los protocolos de red (como HTTP) y hacen uso de XML, para enviar datos compuestos.

Los Servicios Web basados en REST han adoptado el propio funcionamiento de la Web, siendo realmente simple transferir la información. Disponen de unas funciones básicas, para poder interactuar, y así evitar introducir demasiada información extra en los mensajes. Los Servicios Web SOAP, establecen unas pautas muy marcadas para que el cliente y el servidor se puedan entender, lo que hace que en ocasiones ralenticen la interacción.

Las grandes compañías de Internet, están tendiendo a paralizar el soporte de sus Servicios Web SOAP, pasando a potenciar y desarrollar APIs sobre REST (es el caso de Google, Yahoo!, Amazon, etc.). En el marco actual, la facilidad y sencillez de estos servicios está ganando la batalla a SOAP, pero como se ha indicado, todavía hay características que REST no ha llegado a cubrir. Las extensiones WS-* permiten disponer de funcionalidades en muchos casos necesarias, o al menos interesantes, para por ejemplo implementar mayor seguridad. Por tanto, SOAP y REST pueden coexistir, cada uno orientado a un ámbito: SOAP puede cubrir las necesidades empresariales, donde las empresas pueden tener un mayor control de sus servicios internos (y de su información privada), y REST tendería a usarse en las aplicaciones Web actuales (y composiciones de éstas, como por ejemplo en el caso de los Mashups).

En la literatura ofrecida se puede ver cómo los Servicios Web REST superan completamente a los Servicios Web basados en SOAP en relación al rendimiento. Cuando nos referimos a rendimiento, hablamos por ejemplo del número de peticiones que se pueden tratar en un periodo determinado de tiempo. SOAP no fue concebido con el objetivo de ser eficiente, y para enviar un mensaje básico, incluye demasiados datos adicionales. Hemos intentado revisar cómo se podría solucionar esta desventaja, o al menos mejorar este rendimiento, puesto que hay bastantes iniciativas entorno a este aspecto. Por último detallar, que también existe cada vez más actividad en relación a dotar de características extra a los Servicios Web basados en REST, como podría ser una opción equiparable a WS-Security en SOAP.

En el futuro veremos si alguno de los dos gana la carrera, pero mi opinión personal es que quizás la tendencia sea enfocar los WS SOAP para desarrollos empresariales y dejar REST para la Web. El hecho de que las APIs de las principales compañías sean REST, dictaminará que los desarrollares pretendan instaurar esta metodología como referencia, con el consiguiente “peligro” para SOAP.

7. REFERENCIAS

[1] He H. “What Is Service-Oriented Architecture”. Septiembre, 2003. [online] http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[2] Figura extraída de: http://ischool.tv/news/category/youtube/

[3] Figura creada por: Carles Salas

[4] Schekkerman, J. ”Enterprise Architecture & Service Oriented Architecture (SOA)”. [online] http://www.enterprise-architecture.info/Images/Services%20Oriented%20Enterprise/EA_Service-Oriented-Architecture1.htm

[5] Página Web del World Wide Web Consortium: http://www.w3.org/

[6] Página Web de Oasis: http://www.oasis-open.org/home/index.php

[7] Grupo de trabajo Web Services Architecture: http://www.w3.org/2002/ws/arch/

[8] Página Web de XML-RPC: http://www.xmlrpc.com/

[9] Figura extraída de: http://www.di.uniovi.es/~labra/cursos/Web20/images/VocabServiciosWeb.png

[10] Telefónica “Web Services”. Mayo de 2003. [online] http://empresas.telefonica.es/documentacion/WP_Web_Services.pdf

[11] W3C “Guía Breve de Servicios Web”. Última modificación: 09/01/2008. [online] http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

[12] Página web del grupo WS-Interoperability: http://www.ws-i.org

[13] Web sobre WS-Secure: http://www-106.ibm.com/developerworks/webservices/library/ws-secure

[14] Web sobre BPEL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel

[15] Web sobre WS-Transactions specifications de IBM: http://www.ibm.com/developerworks/webservices/library/ws-transpec

[16] Web sobre WS-Coordination de IBM: http://www-106.ibm.com/developerworks/library/ws-coor

[17] Póster de innoQ sobre los estándares de Servicios Web: http://www.innoq.com/soa/ws-standards/poster/innoQ%20WS-Standards%20Poster%202007-02.pdf

[18] L. Costello, “Building Web Services the REST Way”. [online] http://www.xfront.com/REST-Web-Services.html

[19] Página Web de Roy T. Fielding: http://www.ics.uci.edu/~fielding/

[20] Figura extraída de: http://rails.mimbles.net/wp-content/uploads/2008/01/rest_model.png

[21] “Advantages & Disadvantages of Webservices” http://social.msdn.microsoft.com/Forums/en-US/asmxandxml/thread/435f43a9-ee17-4700-8c9d-d9c3ba57b5ef/]

[22] Pautasso C. “RESTfulWeb Services” Presentación. Abril 2008.

[23] Dubray J. “A Fair Comparison of REST and WS-* using an Architectural Decision Framework: is the Debate Over?” Discusión online, Mayo de 2008. http://www.infoq.com/news/2008/05/rest-vs-ws-star

[24] JavaHispano Podcast - 035 – “Arquitectura SOA y Servicios Web (SOAP/REST)”. Febrero de 2009. [online] http://www.javahispano.org/contenidos.item.action?id=7237760&menuId=JH_PODCASTS

[25] Prescod P. “Roots of the REST/SOAP Debate”. [online] http://prescod.net/rest/rest_vs_soap_overview/

[26] “Google REST search API”: http://blogoscoped.com/archive/2008-04-09-n26.html

[27] “REST, ¿Futuro de SOA?”: http://www.espaciosoa.net/2007/06/13/rest-%C2%BFfuturo-de-soa/

[28] Firma HMAC-SHA1: http://docs.amazonwebservices.com/AmazonSimpleDB/2007-11-07/DeveloperGuide/index.html?HMACAuth.html

[29] Figura extraída de: https://www.strongtech.com/i/images/googleapps.jpg

[30] “High Performance SOA – a Contradiction in Terms?”, Noviembre de 2006: http://www.webservices.org/weblog/patrick_leonard/high_performance_soa_a_contradiction_in_terms

[31] Pandrangi N. “PI : Beef up the performance of synchronous webservices”, Julio de 2007: https://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/6912

[32] Sumra R. “Quality of Service for Web Services—Demystification, Limitations, and Best Practices” http://www.developer.com/services/article.php/2027911

[33] “Performance best practices for Web services”, Mayo de 2004: http://www.soaprpc.com/archives/000020.html

[34] Cohen F. “Discover SOAP encoding's impact on Web service performance”. Marzo de

2003. http://www.ibm.com/developerworks/webservices/library/ws-soapenc/

[35] “Debating the Performance of SOAP, REST, HTTP, and Other Distributed Computing Toolkits”: http://hinchcliffe.org/archive/2005/08/10/1540.aspx

[36] Web del proyecto en Google de WS-Cache: http://code.google.com/p/ws-cache/

[37] Pautaso C., Zimmermann O., Leymann F. “RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision”. WWW 2008, April 21–25, 2008, Beijing, China. [online] http://www.jopera.org/files/www2008-restws-pautasso-zimmermann-leymann.pdf