

*THE CONSULTANTS HANDBOOK PART 2:  
MAKE SURE YOUR CUSTOMER IS ALWAYS AGILE*

*geek and poke*

## Índice

¿Qué es un Servicio Web?.....	3
¿Qué es REST realmente? .....	4
¿Cuál es la motivación de REST? .....	4
¿Cuáles son los principios de REST? .....	5
¿Cómo sería un ejemplo de diseño basado en REST?.....	6
¿Como crear una interfaz basada en REST? .....	7
¿Por qué surge el debate entre REST y los Servicios Web? .....	10
¿Por qué surge el debate entre los Servicios Web basados en REST y SOAP?.....	10
¿Cómo diseñar un servicio Web basado en REST? .....	11
¿Cuáles son las características de REST y SOAP en definitiva? .....	12
¿Cuáles son las diferencias? .....	12
¿Es realmente REST la panacea? .....	14
¿Qué puede pasar con SOAP en el futuro?.....	15
¿Qué pasará con REST? .....	16
¿Dónde es útil REST? .....	16
¿Dónde es útil SOAP? .....	17
¿Qué podemos concluir de todo este debate?.....	17

## ¿Qué es un Servicio Web?

El consorcio W3C define los Servicios Web como sistemas software diseñados para soportar una interacción interoperable maquina a maquina sobre una red. Los Servicios Web suelen ser APIs Web que pueden ser accedidas dentro de una red (principalmente Internet) y son ejecutados en el sistema que los aloja.

La definición de Servicios Web propuesta alberga muchos tipos diferentes de sistemas, pero el caso común de uso de refiere a clientes y servidores que se comunican mediante mensajes XML que siguen el estándar SOAP.

En los últimos años se ha popularizado un estilo de arquitectura Software conocido como REST (Representational State Transfer). Este nuevo estilo ha supuesto una nueva opción de estilo de uso de los Servicios Web. A continuación se listan los tres estilos de usos más comunes:

- Remote Procedure Calls (RPC, Llamadas a Procedimientos Remotos): Los Servicios Web basados en RPC presentan una interfaz de llamada a procedimientos y funciones distribuidas, lo cual es familiar a muchos desarrolladores. Típicamente, la unidad básica de este tipo de servicios es la operación WSDL (WSDL es un descriptor del Servicio Web, es decir, el homologo del IDL para COM).

Las primeras herramientas para Servicios Web estaban centradas en esta visión. Algunos lo llaman la primera generación de Servicios Web. Esta es la razón por la que este estilo está muy extendido. Sin embargo, ha sido algunas veces criticado por no ser débilmente acoplado, ya que suele ser implementado por medio del mapeo de servicios directamente a funciones específicas del lenguaje o llamadas a métodos. Muchos especialistas creen que este estilo debe desaparecer.

- Arquitectura Orientada a Servicios (Service-oriented Architecture, SOA). Los Servicios Web pueden también ser implementados siguiendo los conceptos de la arquitectura SOA, donde la unidad básica de comunicación es el mensaje, más que la operación. Esto es típicamente referenciado como servicios orientados a mensajes.

Los Servicios Web basados en SOA son soportados por la mayor parte de desarrolladores de software y analistas. Al contrario que los Servicios Web basados en RPC, este estilo es débilmente acoplado, lo cual es preferible ya que se centra en el “contrato” proporcionado por el documento WSDL, más que en los detalles de implementación subyacentes.

- REST (REpresentation State Transfer). Los Servicios Web basados en REST intentan emular al protocolo HTTP o protocolos similares mediante la restricción de establecer la interfaz a un conjunto conocido de operaciones

estándar (por ejemplo GET, PUT,...). Por tanto, este estilo se centra más en interactuar con recursos con estado, que con mensajes y operaciones.

## ¿Qué es REST realmente?

REST (Representational State Transfer) es un estilo de arquitectura de software para sistemas hipermedias distribuidos tales como la Web. El término fue introducido en la tesis doctoral de Roy Fielding en 2000, quien es uno de los principales autores de la especificación de HTTP.

En realidad, REST se refiere estrictamente a una colección de principios para el diseño de arquitecturas en red. Estos principios resumen como los recursos son definidos y diseccionados. El término frecuentemente es utilizado en el sentido de describir a cualquier interfaz que transmite datos específicos de un domino sobre HTTP sin una capa adicional, como hace SOAP. Estos dos significados pueden chocar o incluso solaparse. Es posible diseñar un sistema software de gran tamaño de acuerdo con la arquitectura propuesta por Fielding sin utilizar HTTP o sin interactuar con la Web. Así como también es posible diseñar una simple interfaz XML+HTTP que no sigue los principios REST, y en cambio seguir un modelo RPC.

Cabe destacar que REST no es un estándar, ya que es tan solo un estilo de arquitectura. Aunque REST no es un estándar, está basado en estándares:

- HTTP
- URL
- Representación de los recursos: XML/HTML/GIF/JPEG/...
- Tipos MIME: text/xml, text/html, ...

## ¿Cuál es la motivación de REST?

La motivación de REST es la de capturar las características de la Web que la han hecho tan exitosa.

Si pensamos un poco en este éxito, nos daremos cuenta que la Web ha sido la única aplicación distribuida que ha conseguido ser escalable al tamaño de Internet. El éxito lo debe al uso de formatos de mensaje extensibles y estándares, pero además cabe destacar que posee un esquema de direccionamiento global (estándar y extensible a su vez).

En particular, el concepto central de la Web es un espacio de URIs unificado. Las URIs permiten la densa red de enlaces que permiten a la Web que sea tan utilizada. Por tanto, ellos consiguen tejer una mega-aplicación.

Las URIs identifican recursos, los cuales son objetos conceptuales. La representación de tales objetos se distribuye por medio de mensajes a través de la Web. Este sistema es extremadamente desacoplado.

Estas características son las que han motivado para ser utilizadas como guía para la evolución de la Web.

## ¿Cuáles son los principios de REST?

El estilo de arquitectura subyacente a la Web es el modelo REST. Los objetivos de este estilo de arquitectura se listan a continuación:

- Escalabilidad de la interacción con los componentes. La Web ha crecido exponencialmente sin degradar su rendimiento. Una prueba de ellos es la variedad de clientes que pueden acceder a través de la Web: estaciones de trabajo, sistemas industriales, dispositivos móviles,...
- Generalidad de interfaces. Gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial. Esto no es del todo cierto para otras alternativas, como SOAP para los Servicios Web.
- Puesta en funcionamiento independiente. Este hecho es una realidad que debe tratarse cuando se trabaja en Internet. Los clientes y servidores pueden ser puestos en funcionamiento durante años. Por tanto, los servidores antiguos deben ser capaces de entenderse con clientes actuales y viceversa. Diseñar un protocolo que permita este tipo de características resulta muy complicado. HTTP permite la extensibilidad mediante el uso de las cabeceras, a través de las URIs, a través de la habilidad para crear nuevos métodos y tipos de contenido.
- Compatibilidad con componentes intermedios. Los más populares intermediarios son varios tipos de proxys para Web. Algunos de ellos, las caches, se utilizan para mejorar el rendimiento. Otros permiten reforzar las políticas de seguridad: firewalls. Y por último, otro tipo importante de intermediarios, gateway, permiten encapsular sistemas no propiamente Web. Por tanto, la compatibilidad con intermediarios nos permite reducir la latencia de interacción, reforzar la seguridad y encapsular otros sistemas.

REST logra satisfacer estos objetivos aplicando cuatro restricciones:

- Identificación de recursos y manipulación de ellos a través de representaciones. Esto se consigue mediante el uso de URIs. HTTP es un protocolo centrado en URIs. Los recursos son los objetos lógicos a los que se le envían mensajes. Los recursos no pueden ser directamente accedidos o modificados. Más bien se trabaja con representaciones de ellos. Cuando se utiliza un método PUT para enviar información, se coge como una representación de lo que nos gustaría que

el estado del recurso fuera. Internamente el estado del recurso puede ser cualquier cosa desde una base de datos relacional a un fichero de texto.

- Mensajes autodescriptivos. REST dicta que los mensajes HTTP deberían ser tan descriptivos como sea posible. Esto hace posible que los intermediarios interpreten los mensajes y ejecuten servicios en nombre del usuario. Uno de los modos que HTTP logra esto es por medio del uso de varios métodos estándares, muchos encabezamientos y un mecanismo de direccionamiento. Por ejemplo, las cachés Web saben que por defecto el comando GET es cacheable (ya que es side-effect-free) en cambio POST no lo es. Además saben como consultar las cabeceras para controlar la caducidad de la información. HTTP es un protocolo sin estado y cuando se utiliza adecuadamente, es posible interpretar cada mensaje sin ningún conocimiento de los mensajes precedentes. Por ejemplo, en vez de logearse del modo que lo hace el protocolo FTP, HTTP envía esta información en cada mensaje.
- Hipertexto como un mecanismo del estado de la aplicación. El estado actual de una aplicación Web debería ser capturada en uno o más documentos de hipertexto, residiendo tanto en el cliente como en el servidor. El servidor conoce sobre el estado de sus recursos, aunque no intenta seguirle la pista a las sesiones individuales de los clientes. Esta es la misión del navegador, el sabe como navegar de recurso a recurso, recogiendo información que el necesita o cambiar el estado que el necesita cambiar.

En la actualidad existen millones de aplicaciones Web que implícitamente heredan estas restricciones de HTTP. Hay una disciplina detrás del diseño de sitios Web escalables que puede ser aprendida de los documentos de arquitectura Web o de varios estándares. Por otra parte, también es verdad que muchos sitios Web comprometen uno más de estos principios, como por ejemplo, seguir la pista de los usuarios moviéndose a través de un sitio. Esto es posible dentro de la infraestructura de la Web, pero daña la escalabilidad, volviendo un medio sin conexión en todo lo contrario.

Los defensores de REST han creído que estas ideas son tan aplicables a los problemas de integración de aplicaciones como los problemas de integración de hipertexto. Fielding es bastante claro diciendo que REST no es la cura para todo. Algunas de estas características de diseño no serán apropiadas para otras aplicaciones. Sin embargo, aquellos que han decidido adoptar REST como un modelo de servicio Web sienten que al menos articula una filosofía de diseño con fortaleza, debilidades y áreas de aplicabilidad documentada.

## ¿Cómo sería un ejemplo de diseño basado en REST?

De nuevo tomaremos como ejemplo a la Web. La Web evidentemente es un ejemplo clave de diseño basado en REST, ya que muchos principios son la base de REST. Posteriormente mostraremos un posible ejemplo real aplicado a Servicios Web.

La Web consiste del protocolo HTTP, de tipos de contenido, incluyendo HTML y otras tecnologías tales como el Domain Name System (DNS).

Por otra parte, HTML puede incluir javascript y applets, los cuales dan soporte al code-on-demand, y además tiene implícitamente soporte a los vínculos. HTTP posee un interfaz uniforme para acceso a los recursos, el cual consiste de URIs, métodos, códigos de estado, cabeceras y un contenido guiado por tipos MIME.

Los métodos HTTP más importantes son PUT, GET, POST y DELETE. Ellos suelen ser comparados con las operaciones asociadas a la tecnología de base de datos, operaciones CRUD: CREATE, READ, UPDATE, DELETE. Otras analogías pueden también ser hechas como con el concepto de copiar-y-pegar (Copy&Paste). Todas las analogías se representan en la siguiente tabla:

<b>Acción</b>	<b>HTTP</b>	<b>SQL</b>	<b>Copy&amp;Paste</b>	<b>Unix Shell</b>
Create	PUT	Insert	Pegar	>
Read	GET	Select	Copiar	<
Update	POST	Update	Pegar después	>>
Delete	DELETE	Delete	Cortar	Del/rm

Las acciones (verbos) CRUD se diseñaron para operar con datos atómicos dentro del contexto de una transacción con la base de datos. REST se diseña alrededor de transferencias atómicas de un estado más complejo, tal que puede ser visto como la transferencia de un documento estructurado de una aplicación a otra.

El protocolo HTTP separa las nociones de un servidor y un navegador. Esto permite a la implementación cada uno variar uno del otro, basándose en el concepto cliente/servidor. Cuando utilizamos REST, HTTP no tiene estado. Cada mensaje contiene toda la información necesaria para comprender la petición cuando se combina el estado en el recurso. Como resultado, ni el cliente ni el servidor necesita mantener ningún estado en la comunicación. Cualquier estado mantenido por el servidor debe ser modelado como un recurso.

La restricción de no mantener el estado puede ser violada mediante cookies que mantienen las sesiones. Fielding advierte del riesgo a la privacidad y seguridad que frecuentemente surge del uso de cookies, así como la confusión y errores que pueden resultar de las interacciones entre cookies y el uso del boton “Go back” del navegador.

HTTP proporciona mecanismos para el control del caching y permite que ocurra una conversación entre el navegador y la caché del mismo modo que se hace entre el navegador y el servidor Web.

## ¿Como crear una interfaz basada en REST?

En vez de cubrir esto desde un punto de vista arquitectural, es aconsejable realizarlo a modo de receta. Existen una serie de pasos a tomar y una serie de preguntas a responder.

Antes de empezar a pensar en el servicio, debemos responder las siguientes preguntas en orden:

- ¿Qué son las URIs? Las cosas identificadas por URIs son “recursos”. Aunque es más apropiado decir que los Recursos son identificados mediante URIs.

Si solo existe una única URI como punto de acceso, posiblemente se esté creando un protocolo RPC. En tal caso, se debe desmenuzar el problema en tipos de recursos que se quieren manipular. Dos lugares donde se debe considerar cuando se buscan los recursos potenciales son las colecciones y las interfaces de búsqueda. Una colección de recursos es en si mismo un recurso. Una interfaz de búsqueda también lo es, ya que el resultado de una búsqueda es otro conjunto de recursos.

A modo de ejemplo, supongamos un sistema de mantenimiento de una lista de contactos de empleados. En tal sistema cada usuario debería tener su propia URI con una apropiada representación. Además, la colección de recursos es otro recurso.

Por tanto, hemos identificado dos tipos de recursos, por tanto habrá dos tipos de URIs:

- Employee (Una URI por empleado).
  - AllEmployee (Listado de los empleados).
- ¿Cuál es el formato? Cuando hablamos informalmente de formato, estamos hablando de la representación. Como ya hemos comentado con anterioridad, no se puede acceder directamente al recurso, hay que obtener una representación. Esta representación puede ser un documento HTML, XML, una imagen,... dependiendo de la situación.

Para cada uno de los recursos, se tiene que decidir cual va a ser su representación. Si es posible, reutilizar formatos existentes, ayudará a incrementar la oportunidad de que nuestro sistema se componga con otros.

Continuando con nuestro ejemplo, el formato de representación va a ser XML, ya que es el principal formato para intercambio de información. El formato del empleado podría ser el siguiente:

```
<employee xmlns='HTTP://example.org/my-example-ns/'>
  <name>Full name goes here.</name>
  <title>Persons title goes here.</title>
  <phone-number>Phone number goes here.</phone-number>
</employee>
```

Para el listado de empleados podríamos tomar este otro:

```
<employee-list xmlns='HTTP://example.org/my-example-ns/'>
  <employee-ref href="URI of the first employee"/>
    Full name of the first employee goes here.</employee>
```



```

    <employee-ref href="URI of employee #2"/>Full
name</employee>
.
.
    <employee-ref href="URI of employee #N"/>Full
name</employee>
</employee-list>

```

- ¿Qué métodos son soportados en cada URI? En este apartado tenemos que definir como van a ser referenciados los recursos. Por medio de las URIs y los métodos soportados el acceso va a ser posible. El acceso se puede hacer de muchas formas, recibiendo una representación del recurso (GET o HEAD), añadiendo o modificando una representación (POST o PUT) y eliminando algunas o todas las representaciones (DELETE).

HTTP	CRUD	Descripción
POST	CREATE	Crear un nuevo recurso
GET	RETRIEVE	Obtener la representación de un recurso
PUT	UPDATE	Actualizar un recurso
DELETE	DELETE	Eliminar un recurso

Continuando con nuestro ejemplo, ahora vamos a combinar recursos, representación y métodos:

Recurso	Método	Representación
Employee	GET	Formato del empleado
Employee	PUT	Formato del empleado
Employee	DELETE	-
All Employees	GET	Formato de la lista de empleados
All Employees	POST	Formato de empleado

- ¿Qué códigos de estado pueden ser devueltos? No solo es necesario conocer que tipo de representación va a ser devuelta, también es necesario enumerar los códigos de estado HTTP típicos que podrían ser devueltos.

Volvemos a actualizar nuestra tabla para mostrar los códigos de estado:

Recurso	Método	Representación	Códigos de estado
Employee	GET	Formato del empleado	200, 301, 410
Employee	PUT	Formato del empleado	200, 301, 400, 410
Employee	DELETE	-	200, 204
All Employees	GET	Formato de la lista de empleados	200, 301
All Employees	POST	Formato de empleado	201, 400

## **¿Por qué surge el debate entre REST y los Servicios Web?**

Esta pregunta es un tanto errónea, aunque es bastante típica en los foros de discusión. REST es un estilo, mientras que los servicios Web son sistemas software. Por tanto, no es posible la comparación de ambos conceptos. Por otra parte, popularmente se generaliza el concepto de servicio Web con el de servicio Web basado en SOAP. Como hemos visto en apartados anteriores, es posible diseñar servicios Web basados en REST, es decir tomando REST como estilo de diseño.

Por tanto, esta pregunta debería haber sido formulada como ha sido hecha en el siguiente apartado.

## **¿Por qué surge el debate entre los Servicios Web basados en REST y SOAP?**

Muchos diseñadores de Servicios Web están llegando a la conclusión que SOAP es demasiado complicado. Por tanto, están comenzando a utilizar Servicios Web basados en REST para mostrar cantidades de datos masivos. Este es el caso de grandes empresas como eBay y Google.

El problema principal surge del propósito inicial de SOAP. Esta tecnología fue originalmente pensada para ser una versión, sobre Internet, de DCOM o CORBA. Así lo demuestra su predecesor, el protocolo XML-RPC. Estas tecnologías lograron un éxito limitado antes de ser adaptadas. Esto es debido a que este tipo de tecnologías, las basadas en modelos RPC (Remote Procedure Call) son más adecuadas para entornos aislados, es decir, entornos donde se conoce perfectamente el entorno. La evolución en este tipo de sistemas es sencilla, se modifica cada usuario para que cumpla con los nuevos requisitos.

Pero cuando el número de usuarios es muy grande es necesario emplear una estrategia diferente. Se necesita organizar frameworks que permitan evolucionar, tanto por el lado del cliente como del servidor. Se necesita proponer un mecanismo explícito para la interoperabilidad de los sistemas que no poseen la misma API. Protocolos basados en RPC no son adecuados para este tipo de sistemas, ya que cambiar su interfaz resulta complicado.

Por esta razón, se intenta tomar como modelo a la Web. A primera vista se puede pensar que SOAP lo hace, ya que utiliza HTTP como medio de transporte. Pero Fielding argumenta que la Web funciona mejor cuando se utiliza en el estilo que lo hace REST. Utilizar HTTP como medio de transporte para protocolos de aplicación a través de firewalls es una idea equivocada. Esto reduce la efectividad de tener un firewall. Lo cual aumenta las posibilidades de nuevos agujeros de seguridad.

Sin embargo, los partidarios de SOAP argumentan que gracias a la tecnología existente que permite a los diseñadores encapsular la complejidad del sistema, dando lugar a interfaces generadas automáticamente que permiten facilitar el diseño del sistema.

Según argumenta Spolsky lo interesante de SOAP es que permite utilizar lenguajes de alto nivel para llamar y para implementar el servicio. Añade que “Alegar que SOAP es malo porque el formato de conexión es horrible y pesado es como alegar que nadie debería utilizar Pentiums porque su juego de instrucciones es mucho más complicado que el juego de instrucciones del 8086. Eso es cierto, pero por esa razón existen los compiladores.”

## ¿Cómo diseñar un servicio Web basado en REST?

Las pautas a seguir en el diseño de servicios Web basados en REST son las mismas que las descritas en el apartado dedicado a crear una interfaz REST. Por otra parte, hemos ampliado dicha información en el contexto de los Servicios Web:

- Identificar todas las entidades conceptuales que se desean exponer como servicio.
- Crear una URL para cada recurso. Los recursos deberían ser nombres no verbos (acciones). Por ejemplo no utilizar esto:

```
http://www.service.com/entities/getEntity?id=001
```

Como podemos observar, `getEntity` es un verbo. Mejor utilizar el estilo REST, un nombre:

```
http://www.service.com/entities/001
```

- Categorizar los recursos de acuerdo con si los clientes pueden obtener una representación del recurso o si pueden modificarlo. Para el primero, debemos hacer los recursos accesibles utilizando un HTTP GET. Para el último, debemos hacer los recursos accesibles mediante HTTP POST, PUT y/o DELETE.
- Todos los recursos accesibles mediante GET no deberían tener efectos secundarios. Es decir, los recursos deberían devolver la representación del recurso. Por tanto, invocar al recurso no debería ser el resultado de modificarlo.
- Ninguna representación debería estar aislada. Es decir, es recomendable poner hipervínculos dentro de la representación de un recurso para permitir a los clientes obtener más información.
- Especificar el formato de los datos de respuesta mediante un esquema (DTD, W3C Schema, ...). Para los servicios que requieran un POST o un PUT es aconsejable también proporcionar un esquema para especificar el formato de la respuesta.
- Describir como nuestro servicio ha de ser invocado, mediante un documento WSDL/WADL o simplemente HTML.

## ¿Cuáles son las características de REST y SOAP en definitiva?

Según hemos visto a lo largo del documento, el principal beneficio de SOAP recae en ser fuertemente acoplado, lo que permite poder ser testado y depurado antes de poner en marcha la aplicación. En cambio, las ventajas de la aproximación basada en REST recaen en la potencial escalabilidad de este tipo de sistemas, así como el acceso con escaso consumo de recursos a sus operaciones debido al limitado número de operaciones y el esquema de direccionamiento unificado.

A modo de resumen, veamos las características de ambas aproximaciones en la siguiente tabla:

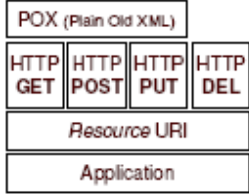
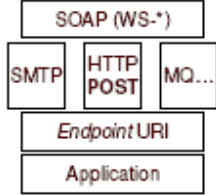
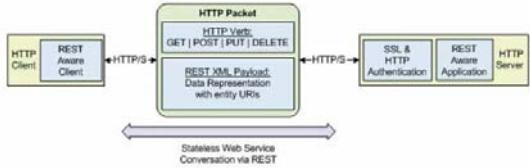
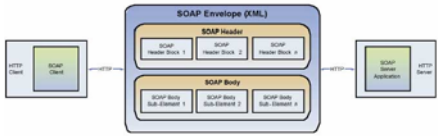
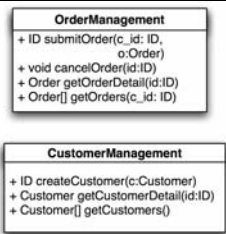
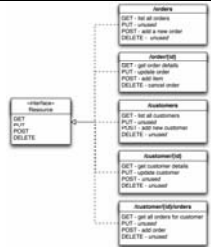
	REST	SOAP
Características	Las operaciones se definen en los mensajes. Una dirección única para cada instancia del proceso. Cada objeto soporta las operaciones estándares definidas. Componentes débilmente acoplados.	Las operaciones son definidas como puertos WSDL. Dirección única para todas las operaciones. Múltiple instancias del proceso comparten la misma operación. Componentes fuertemente acoplados.
Ventajas declaradas	Bajo consumo de recursos. Las instancias del proceso son creadas explícitamente. El cliente no necesita información de enrutamiento a partir de la URI inicial. Los clientes pueden tener una interfaz “listener” (escuchadora) genérica para las notificaciones. Generalmente fácil de construir y adoptar.	Fácil (generalmente) de utilizar. La depuración es posible. Las operaciones complejas pueden ser escondidas detrás de una fachada. Envolver APIs existentes es sencillo Incrementa la privacidad. Herramientas de desarrollo.
Posibles desventajas	Gran número de objetos. Manejar el espacio de nombres (URIs) puede ser engorroso. La descripción sintáctica/semántica muy informal (orientada al usuario). Pocas herramientas de desarrollo.	Los clientes necesitan saber las operaciones y su semántica antes del uso. Los clientes necesitan puertos dedicados para diferentes tipos de notificaciones. Las instancias del proceso son creadas implícitamente.

## ¿Cuáles son las diferencias?

La principal diferencia entre REST y SOAP se resume en los siguientes puntos de vista del propósito de la Web:

REST	SOAP
“La Web es el universo de la información accesible globalmente” (Tim Berners Lee)	“La Web es el transporte universal de mensajes”

A continuación vamos a intentar esbozar las diferencias entre REST y SOAP desde varios puntos de vista:

	REST	SOAP
Tecnología	Interacción dirigida por el usuario por medio de formularios.	Flujo de eventos orquestados.
	Pocas operaciones con muchos recursos	Muchas operaciones con pocos recursos.
	Mecanismo consistente de nombrado de recursos (URI).	Falta de un mecanismo de nombrado.
	Se centra en la escalabilidad y rendimiento a gran escala para sistemas distribuidos hipermedia.	Se centra en el diseño de aplicaciones distribuidas.
Protocolo		
	 <p>Stateless Web Service Conversation via REST</p>	
	XML autodescriptivo.	Tipado fuerte, XML Schema.
	HTTP.	Independiente del transporte.
	HTTP es un protocolo de aplicación.	HTTP es un protocolo de transporte.
	Síncrono.	Síncrono y Asíncrono.
Descripción del servicio	Confía en documentos orientados al usuario que define las direcciones de petición y las respuestas.	WSDL.
	Interactuar con el servicio supone horas de testado y depuración de URIs.	Se pueden construir automáticamente stubs (clientes) por medio del WSDL.
	No es necesario el tipado fuerte, si ambos lados están de acuerdo con el contenido.	Tipado fuerte.
	WADL propuesto en noviembre de 2006.	WSDL 2.0.
Gestión del estado	El servidor no tiene estado (stateless).	El servidor puede mantener el estado de la conversación.
	Los recursos contienen datos y enlaces representando transiciones a estados válidos.	Los mensajes solo contienen datos.
	Los clientes mantienen el estado siguiendo los enlaces.	Los clientes mantienen el estado suponiendo el estado del servicio.
	Técnicas para añadir sesiones: Cookies	Técnicas para añadir sesiones: Cabecera de sesión (no estándar)
Seguridad	HTTPS.	WS-Security.
	Implementado desde hace muchos años.	Las implementaciones están comenzando a aparecer ahora.
	Comunicación punto a punto segura.	Comunicación origen a destino segura.
Metodología de diseño		
	Identificar recursos a ser expuestos como servicios.	Listar las operaciones del servicio en el documento WSDL.
	Definir URLs para direccionarlos.	Definir un modelo de datos para el contenido de los mensajes.
	Distinguir los recursos de solo lectura (GET) de los modificables (POST, PUT, DELETE).	Elegir un protocolo de transporte apropiado y definir las correspondientes políticas QoS, de seguridad y

		transaccional.
	Implementar e implantar el servidor Web.	Implementar e implantar el contenedor del servicio Web.

## ¿Es realmente REST la panacea?

Evidentemente no. Existen una serie de puntos que resaltan los partidarios de SOAP sobre REST:

- **Asincronía:** Esto normalmente tiene diferentes significados, pero el concepto más común es que dos programas en comunicación (cliente y servidor, en nuestro caso) deberían no necesitar estar en comunicación constante mientras uno de ellos esta realizando una operación costosa. Deberían existir algún tipo de mecanismo mediante el cual un participante pueda avisar al otro cuando el resultado este listo. SOAP no soporta directamente esto, sin embargo puede utilizarse de una manera asíncrona por medio del uso de un transporte asíncrono, aunque esto no está todavía estandarizado. Por otra parte, existen una variedad de propuestas para hacer asíncrono a HTTP, ya que existen una variedad de aspectos a asincronizar. Una de tales especificaciones es conocida como “HTTPEvents”, la cual está propuesta para su estandarización.
- **Routing (Enrutamiento):** Los mensajes HTTP son enrutados del cliente a los proxys y a los servidores. Esto es una especie de enrutamiento controlado por la red, pero a veces es adecuado poder controlar por el cliente el enrutamiento de los mensajes mediante la definición de una ruta entre los nodos. Los partidarios de REST creen que esto puede ser uno de los pocos puntos en común con SOAP, ya que el modo que SOAP utiliza el “Routing” es compatible con la Web. Por tanto, los investigadores de REST trabajan en un modelo similar al de SOAP.
- **Fiabilidad:** Este concepto al igual que el de asincronía puede tener varias interpretaciones. El más común es el envío de una única vez del mensaje. Esto es relativamente fácil de lograr en HTTP, pero no es una característica integrada. La solución consiste en enviar repetidamente e internamente el mensaje hasta obtener una confirmación. Pero el problema recae en el uso del comando POST, recordemos que no es free-side-effect. Por tanto, es necesario incluir en la cabecera algún tipo de identificador del mensaje. En el caso de que el destinatario obtenga un mensaje previamente analizado, el mensaje se desecha.
- **Seguridad:** Los defensores de SOAP argumentan que REST no dispone de mecanismos tan completos de seguridad como es el caso de la especificación WS-Security. Aunque en realidad, esto debería ser estudiado más profundamente.

Una manera muy sencilla de controlar la seguridad consiste en utilizar listas de control de acceso (ACLs). Esta medida de seguridad se puede tomar tanto a nivel global como a nivel local (URIs). Este sistema resulta mucho más difícil de implementar en soluciones basadas en RPC, ya que el sistema de

direccionamiento es propietario y expresado en parámetros arbitrarios. Por otra parte, es posible utilizar las características de seguridad implícitas de HTTP: HTTPS y el sistema de autorización y autenticación de HTTP.

- **Modelo extensible:** SOAP tiene un modelo extensible que permite al creador del mensaje SOAP ser muy explícito sobre si comprender una porción del mensaje es opcional u obligatorio. Esto también permite al encabezamiento ser objetivo de algunos intermediarios. Existe una extensión de HTTP con muchas de las mismas ideas (posiblemente la versión SOAP este basada en esta versión), pero no es tan conocida ni está tan clara sintácticamente.
- **Descripción del servicio:** SOAP tiene WSDL, pero muchos alegan que HTTP no tiene nada similar hasta la fecha. Esto no es del todo cierto, WADL (Web Application Description Language) proporciona una simple alternativa a WSDL para el uso con aplicaciones Web basadas en XML/HTTP. Hasta el momento tales aplicaciones han sido descritas principalmente por medio de combinaciones de descripciones textuales y esquemas XML. WADL pretende proporcionar una descripción de estas aplicaciones procesable automáticamente.
- **Familiaridad:** REST requiere repensar el problema en términos de manipulación de recursos direccionables en vez de llamadas a métodos. Evidentemente, la parte del servidor puede implementarse como se quiera, pero la interfaz con los clientes debe hacerse en términos de REST.

Los clientes podrían preferir una interfaz basada en componente a una interfaz REST. Así como los programadores que están más familiarizados con el uso de APIs. Además, las APIs se integran mucho mejor en los lenguajes de programación existentes. Para los programadores por el lado del cliente, REST puede resultar algo novedoso, en cambio para el otro lado, no se notara mucha diferencia con lo que se había desarrollado en los últimos años, sitios Webs.

## ¿Qué puede pasar con SOAP en el futuro?

El problema de la escalabilidad ya ha sido analizado a lo largo de este documento. Resulta muy fácil de implantar un protocolo dentro de una compañía, pero cuando hablamos de entornos que atraviesan estas fronteras (miles de sistemas), no existe la oportunidad de actualización uno a uno.

Por tanto, las nuevas aplicaciones basadas en SOAP tendrán un gran obstáculo a superar antes de ser implantadas y tendrán incluso mayores retos adaptando y evolucionando una vez hayan sido implantadas.

Por otra parte, posiblemente tendrá varios años de éxito dentro de las organizaciones. Muchos profesionales ven a SOAP como una versión estandarizada de DCOM y CORBA, y por tanto, tendrá al menos el mismo éxito que tuvieron estas tecnologías en la integración punto a punto de sistemas internos. Sin embargo, si una alternativa basada

en REST llega a ser dominante en Internet, será inevitable su filtración a los sistemas corporativos internos como hizo la Web.

## ¿Qué pasará con REST?

Los negocios electrónicos van a necesitar algo más que tecnologías orientadas en RPC. Todos los negocios de cualquier lugar tendrán que estandarizar sus modelos de direccionamiento para exponer las interfaces en común a sus socios. SOAP no permite esto en si mismo, incluso confunde más que aclara. Para que los negocios interoperen sin programar manualmente de manera explícita enlaces a los socios, se necesitará estandarizar un modelo de direccionamiento, más que invertir en sistemas propietarios. REST proporciona un alto grado de estandarización. Por tanto, si los servicios Web basados en SOAP no consiguen implantar este mecanismo, no sobrevivirán y, por tanto, surgirá la era de los Servicios Web basados en REST.

## ¿Dónde es útil REST?

Tanto los arquitectos como los desarrolladores necesitan decidir cual es el estilo adecuado para las aplicaciones. En algunos casos es adecuado un diseño basado en REST, se listan a continuación:

- El servicio Web no tiene estado. Una buena comprobación de esto consistiría en considerar si la interacción puede sobrevivir a un reinicio del servidor.
- Una infraestructura de caching puede mejorar el rendimiento. Si los datos que el servicio Web devuelve no son dinámicamente generados y pueden ser cacheados, entonces la infraestructura de caching que los servidores Web y los intermediarios proporcionan, pueden incrementar el rendimiento.
- Tanto el productor como el consumidor del servicio conocen el contexto y contenido que va a ser comunicado. Ya que REST no posee todavía (aunque hayamos visto una propuesta interesante) un modo estándar y formal de describir la interfaz de los servicios Web, ambas partes deben estar de acuerdo en el modo de intercambiar de información.
- El ancho de banda es importante y necesita ser limitado. REST es particularmente útil en dispositivos con escasos recursos como PDAs o teléfonos móviles, donde la sobrecarga de las cabeceras y capas adicionales de los elementos SOAP debe ser restringida.
- La distribución de Servicios Web o la agregación con sitios Web existentes puede ser fácilmente desarrollada mediante REST. Los desarrolladores pueden utilizar tecnologías como AJAX y toolkits como DWR (Direct Web Remoting) para consumir el servicio en sus aplicaciones Web.



## ¿Dónde es útil SOAP?

Un diseño basado en SOAP es adecuado cuando:

- Se establece un contrato formal para la descripción de la interfaz que el servicio ofrece. El lenguaje de Descripción de Servicios Web (WSDL), como ya sabemos, permite describir con detalles el servicio Web.
- La arquitectura debe abordar requerimientos complejos no funcionales. Muchas especificaciones de servicios Web abordan tales requisitos y establecen un vocabulario común para ellos. Algunos ejemplos incluyen transacciones, seguridad, direccionamiento, ... La mayoría de aplicaciones del mundo real se comportan por encima de las operaciones CRUD y requieren mantener información contextual y el estado conversacional. Con la aproximación REST, abordar este tipo de arquitecturas resulta más complicado.
- La arquitectura necesita manejar procesado asíncrono e invocación. En estos casos, la infraestructura proporcionada por estándares como WSRM y APIs como JAX-WS junto con la asincronía por el lado del cliente nos permitirán el soporte de estas características.

## ¿Qué podemos concluir de todo este debate?

Amazon posee ambos estilos de uso de sus servicios Web. Pero el 85% de sus clientes prefieren la interfaz REST. A pesar de la promoción que las empresas han invertido para ensalzar a SOAP, parece que es evidente que los desarrolladores prefieren, en algunos casos, la aproximación más sencilla: REST.

Todo lo visto a lo largo del documento como en el párrafo anterior, parece que nos da pistas para el futuro éxito de REST. Aunque, todos sabemos que en el mundo de la tecnología no siempre acaba triunfando la tecnología mejor, recuérdese el caso de VHS vs BetaMax.

Lo que está claro es que falta una pieza en la Web. La comunicación Hombre – Máquina parece que funciona, pero la comunicación Máquina – Máquina sigue siendo un reto. Recientemente, se ha presentado una propuesta donde los principios de REST han sido aplicados a los estándar y guías de diseño asociadas con la nueva versión de SOAP, es decir, SOAP podría ser utilizado de tal manera que no violara los principios de REST. Esto parece prometedor. Pero en mi opinión, este tipo de propuestas (incluyendo a REST) no triunfarán si la industria no apuesta realmente por ellas (herramientas y frameworks).

## Referencias

- William Brogden, "REST versus SOAP – the REST story", [http://searchwebservices.techtarget.com/tip/0,289483,sid26\\_gci1227190,00.html](http://searchwebservices.techtarget.com/tip/0,289483,sid26_gci1227190,00.html)
- William Brogden, "REST versus SOAP – the SOAP story", [http://searchwebservices.techtarget.com/tip/0,289483,sid26\\_gci1231889,00.html](http://searchwebservices.techtarget.com/tip/0,289483,sid26_gci1231889,00.html)
- Roger L. Costello, "Building Web Services the REST Way". xFront website. <http://www.xfront.com/REST-Web-Services.html>
- Roy T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", PhD thesis, UC Irvine, 2000, <http://roy.gbiv.com/pubs/dissertation/top.htm>
- Joe Gregorio, "How to create a REST Protocol". XML.com website. <http://www.xml.com/pub/a/2004/12/01/restful-web.html>
- Marc Hadley, "Introducing WADL", [http://weblogs.java.net/blog/mhadley/archive/2005/05/introducing\\_wad.html](http://weblogs.java.net/blog/mhadley/archive/2005/05/introducing_wad.html)
- M zur Muehlen, JV Nickerson, Keith D Swenson, "Developing Web Services Choreography Standards-The Case of REST vs. SOAP", Decision Support Systems. Vol. 40, no. 1, pp. 9-29. July 2005, ISSN:0167-9236 <http://www.workflowresearch.com/Publications/PDF/MIZU.JENI.KESW-DSS%282004%29.pdf>
- Tim O'Reilly, "REST vs SOAP at Amazon", O'Reilly's XML.com website, <http://www.oreillynet.com/pub/wlg/3005>
- Cesare Pautasso, "SOAP vs REST. Bringing the Web back into Web Services". IBM Zurich Research Lab. [http://www.iks.inf.ethz.ch/education/ss07/ws\\_soa/slides/SOAPvsREST\\_ETH.pdf](http://www.iks.inf.ethz.ch/education/ss07/ws_soa/slides/SOAPvsREST_ETH.pdf)
- Paul Prescod, "Second Generation Web Services", O'Reilly's XML.com website, February 06, 2002, <http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html>
- Paul Prescod, "Roots of the REST/SOAP Debate". [http://www.prescod.net/rest/rest\\_vs\\_soap\\_overview/](http://www.prescod.net/rest/rest_vs_soap_overview/)
- Paul Prescod, "REST and the real world". O'Reilly's XML.com website, <http://www.xml.com/lpt/a/923>
- Leonard Richardson and Sam Ruby, "RESTful Web Services", O'Reilly Publishers, 2007.
- Daniel Rubio, "WADL: The REST answer to WSDL". [http://searchwebservices.techtarget.com/tip/0,289483,sid26\\_gci1265367,00.html](http://searchwebservices.techtarget.com/tip/0,289483,sid26_gci1265367,00.html)
- Rich Seeley, "Burton sees the future of SOA and it is REST". [http://searchwebservices.techtarget.com/originalContent/0,289142,sid26\\_gci1256796,00.html](http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci1256796,00.html)
- RestWiki, "RestWiki". <http://rest.blueoxen.net/cgi-bin/wiki.pl?FrontPage>
- Stefan Tilkov, "REST vs SOAP. Oh no, Not Again". [http://www.innoq.com/blog/st/2006/06/30/rest\\_vs\\_soap\\_oh\\_no\\_not\\_again.html](http://www.innoq.com/blog/st/2006/06/30/rest_vs_soap_oh_no_not_again.html)
- Sameer Tyagi, "RESTful Web Services", <http://java.sun.com/developer/technicalArticles/WebServices/restful/>

- Wikipedia, “Representational State Transfer”, [http://en.wikipedia.org/wiki/Representational\\_State\\_Transfer](http://en.wikipedia.org/wiki/Representational_State_Transfer)
- Wikipedia, “Web Services”, [http://en.wikipedia.org/wiki/Web\\_service](http://en.wikipedia.org/wiki/Web_service)