REST – REpresentational State Transfer

Javier González Pisano

Programa de Doctorado: Avances en Informática (2006-2007).
Universidad de Oviedo
Curso de Tecnologías, Estándares y Servicios Web.
jgpisano@gmail.com

Resumen

Desde la aparición de los servicios Web han surgido diversas críticas debido a las limitaciones de éstos, algunas solventadas mediante la nueva generación y otras mediante la llamada Web Semántica. REST intenta abordar la resolución de dichos problemas dando un enfoque distinto al usado anteriormente. Este artículo pretende dar una visión general sobre la arquitectura REST, describiéndola en primer lugar, viendo luego como se relaciona con la Web y con estándares como SOAP y WSDL y terminando con un breve análisis práctico de la cuestión.

Palabras Clave: Servicios Web, recursos, SOAP, URI, transferencia de estado.

1. Introducción a REST

REST es el acrónimo de **Transferencia de Estado Representacional**, término usado por Roy Fielding (uno de los creadores de HTTP) para describir un estilo de arquitectura que utilizar como modelo en los sistemas de computación Web. No es un estándar, sino un enfoque que muestra como desarrollar y proporcionar servicios en Internet, por tanto considerado como un estilo arquitectónico para diseño de software a gran escala. El propio autor lo define de manera concisa en éste párrafo[1]:

"*REST es un intento de mostrar cómo debe comportarse una aplicación Web bien diseñada: una red de páginas Web (una máquina de estados virtual) donde el usuario progresará seleccionando enlaces (transiciones de estado) que devuelven la página siguiente (el siguiente estado de la máquina) que el usuario manejará a su gusto*".

Éste enfoque no aporta realmente nada nuevo: la mayor parte de la Web sigue este estilo arquitectónico. Sin embargo, el autor hace énfasis en las características concretas que han hecho que la Web triunfe, que van a ser los objetivos buscados por REST y que son, a su juicio:

- Escalabilidad en las interacciones entre componentes: La red ha ido creciendo de manera exponencial y se ha ido comportado de manera satisfactoria.
- Generalidad en las interfaces: Se proporciona acceso con distintos clientes y con distintos mecanismos de acceso.
- Desarrollo independiente de componentes: Implementaciones cliente y servidor pueden ser desarrolladas en distintos momentos.
- Existencia de componentes intermediarios con los cuales existe compatibilidad (como proxys) que permiten encapsulación e integración de sistemas no Web dentro de la misma.

1.1. Principios de diseño

Los objetivos citados van a ser conseguidos imponiendo una serie de restricciones:

- El estado y la funcionalidad de las aplicaciones se representan por forma de recursos.
- La identificación de recursos se realiza de forma única global mediante *Uniform Resource Identifiers*. Los recursos identificados con URIs son los objetos lógicos a los cuales se les mandan los mensajes.
- Manipulación de recursos a través de representaciones, luego no se manejan directamente los recursos, sino sus representaciones.
- Todos los recursos comparten un interfaz uniforme formado por:
    - Un conjunto de operaciones limitado para transferir el estado. Se van a aprovechar las operaciones que HTTP define, mostradas en la Fig. 1:

MÉTODO	FUNCIÓN
GET	Solicitar recurso
POST	Crear recurso nuevo
PUT	Actualizar o modificar recurso
DELETE	Borrar recurso

Fig. 1: Operaciones definidas por HTTP

- Un conjunto limitado de tipos de contenidos, identificados mediante tipos MIME.

- Uso de un protocolo cliente/servidor, sin estado y basado en capas. Cada mensaje HTTP contendrá la información necesaria para comprender la petición, luego como muestra la Fig. 2, no es necesario que ésta sea entendida tanto en el cliente como en el servidor.

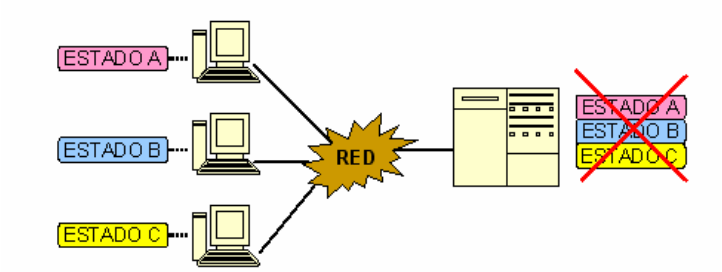

Fig. 2: Almacenamiento del estado en la parte cliente

- Uso de hipermedios para representar el estado de una aplicación, permitiendo que el estado de una aplicación Web particular esté almacenado en uno o más documentos de hipertexto que residen bien en el servidor o en el cliente. Esto permite al servidor saber el estado de sus recursos sin necesidad de almacenar el estado de de los clientes concretos.

Como ya se mencionó, REST se trata de un estilo arquitectónico, no un estándar. Sin embargo hace uso de varios estándares Web:

- HTTP [RFC 1945]: HyperText Transfer Protocol.
- URL [RFC 1738] (*Uniform Resource Locator*) como el mecanismo de identificación de recursos.
- XML / HTML / PNG / etc.. como distintos formatos de representación de recursos.
- Tipos MIME, como text/xml, text/html, image/png, etc..

2. SOAP vs. REST

SOAP [W3C-SOAP][10] y los Servicios Web [8] han revolucionado el panorama de los sistemas distribuidos, usando tecnologías y estándares desarrollados por el W3C [11] (World Wide Web Consortium). Los defensores de REST encuentran a éste modelo una serie de limitaciones que pueden ser superadas adoptando otro enfoque distinto. Como suele pasar en éstos casos, la discusión en ocasiones toma tintes políticos al unir SOAP con Microsoft.

2.1. Diferencia de enfoque

Los Servicios Web SOAP están basados en Remote Procedure Calling (RPC). En RPC, el énfasis se pone en la diversidad de operaciones (o verbos). Así pues, una aplicación puede definir un conjunto de operaciones como las que muestra el Código 1: [9]

```
getUser()          addUser()        remove User()

updateUser()      getLocation()     addLocation()

removeLocation() updateLocation()  listUsers()
```

```
listLocations()  findLocation()    findUser()
```

Código 1: Operaciones definidas con RPC

REST pone el énfasis en la diversidad de recursos (nombres). Para la misma aplicación bastaría con definir dos tipos de recursos:

```
User {}     Location{}
```

Código 2: Recursos definidos con REST

Cada recurso va a tener su propia localización, identificada por un URL (consideremos en éste caso `http://www.example.org/locations/spain/oviedo`). Un registro del recurso Usuario podría tener el siguiente aspecto:

```
<usuario>
 <nombre>Benito Pérez</nombre>
 <genero>masculino</genero>
<localizacion
   href="http://www.example.org/locations/spain/oviedo">
        Oviedo, Spain
</localizacion>
</usuario>
```

Código 3: Detalle de un recurso XML

Los recursos pueden ser cacheados, copiados o marcados. Las operaciones que los clientes pueden realizar vendrán determinadas por las operaciones HTTP estándar, luego un cliente podría solicitar un recurso (como el registro XML anterior) con una petición GET, posteriormente modificarlo y subirlo de nuevo usando una petición PUT.

En contraste con las tecnologías basadas en SOAP, donde todos los nombres de los métodos y convenciones necesitan ser conocidas, los clientes HTTP pueden comunicarse con servidores sin necesidad de realizar ninguna operación de configuración. De este modo se logra la “generalización de interfaces” ya comentada anteriormente.

La parte negativa del enfoque consiste en que HTTP no proporciona ningún método estándar para descubrir recursos (operaciones del estilo FIND o LIST que se corresponderían con las mostradas en el ejemplo RPC). Para solucionarlo, las aplicaciones REST tratan un conjunto de resultados de búsqueda como otro tipo de recurso, requiriendo pues que los diseñadores de aplicaciones conozcan direcciones adicionales para listar las búsquedas de cada tipo de recurso. Así pues, una petición GET sobre la URL anterior podría devolver un enlace a una lista de ficheros en XML con las localizaciones posibles en Oviedo, mientras que la petición GET a `http://www.example.org/users?apellido=Alonso` devolvería una lista de enlaces a todos los usuarios con el apellido "Alonso". La iniciativa OpenSearch [12] de A9.com intenta estandarizar las búsquedas que usan REST estableciendo especificaciones recursos, así como un formato genérico para los sistemas basados en REST.

2.2. Uso de la Web y URLs.

Según los propios creadores de la Web [6], el principal objetivo de ésta es crear un espacio para compartir la información. Los sistemas pueden participar publicando objetos y servicios en el mismo. Ésta idea se refleja en la visión del W3C acerca del futuro arquitectónico de la Web:

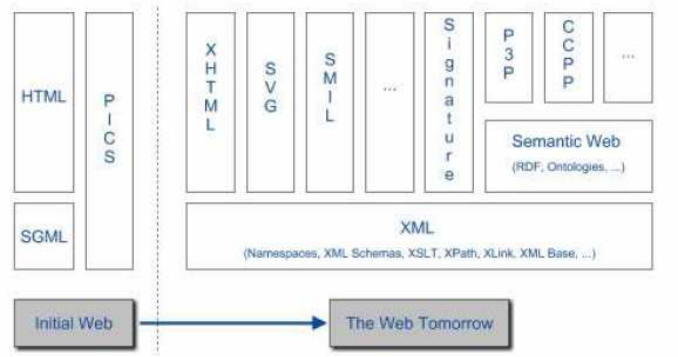

Fig. 3: Futuro de la Web de acorde al W3C.

La terminología usada es muy distinta de la usada con los Servicios Web basados en SOAP, donde es común hablar de la Web como una "transporte" para mensajes que esencialmente son interpretados por sistemas externos. En éste caso, la Web es simplemente un medio de transporte para los sistemas. Éste enfoque es destructivo a la larga, porque dos aplicaciones que se comunican de éste modo no han establecido un diálogo común con las otras dos. Éstos sistemas tampoco han adoptado el modelo Web para el uso de URIs, pues cada aplicación puede establecer su propio espacio de nombres desde cero, en lugar de usar URIs como un mecanismo de direccionamiento. Los estándares no han sido diseñados para usar URIs como identificadores de recursos (cada WSDL describe sólo un recurso Web, siendo imposible describir enlaces a otros recursos).

3. Escenarios aplicables

Las principales ventajas aportadas por REST son: [5]

- Mejores tiempos de respuesta y disminución de carga en servidor.
- Mayor escalabilidad al no requerir mantenimiento de estado.
- Facilita desarrollo de clientes.
- Mayor estabilidad frente a futuros cambios.
- Permite creación independiente de los tipos de documentos, sin afectar a los anteriores.

Dichas ventajas aconsejan su uso en una serie de escenarios localizados:

- Cuando los recursos van a estar disponibles para un grupo de usuarios desconocidos potencialmente grande.
- Cuando el propósito de la aplicación es mandar información.
- Cuando está previsto que la aplicación crezca continuamente.
- Cuando el uso de recursos hardware (CPU, memoria, ancho de banda…) no es determinante.
- Cuando la gestión del estado de la aplicación es simple.

Existen sin embargo una serie de situaciones en las cuales no nos será útil:

- Situaciones donde la seguridad de los datos es determinante.
- Cuando el propósito principal de la aplicación es recibir y procesar grandes cantidades de información.
- Cuando el uso de los recursos hardware es determinante.
- Cuando la gestión del estado de la aplicación no es trivial.

4. Implementaciones públicas

Dado que la definición de REST es muy amplia, es posible afirmar que existe un enorme número de aplicaciones REST en la red (prácticamente cualquier cosa accesible mediante una petición HTTP GET). De forma más restrictiva, y si lo valoramos en contraposición a los Servicios Web y el RPC, REST puede encontrarse en distintas áreas de la Web:

- La mayor parte de los blogs están basados en REST, dado que implica descarga de ficheros XML (en formato RSS o Atom) que contienen listas de enlaces a otros recursos.
- Amazon.com [13] ofrece su interfaz para desarrolladores tanto en formato REST como en formato SOAP (siendo la versión REST la que recibe mayor tráfico). Pionera en el uso de REST, ofrece una base de datos con todos los productos que vende.
- Ebay [14] ofrece un interfaz REST para desarrolladores, permitiendo la consulta de productos a través del método `GetSearchResults()`.
- YoutTube [16] , Yahoo [15] , Flickr [17] u otros también ofrecen un conjunto de interfaces “REST” (algunos de estos sistemas no respetan las restricciones estructurales dadas por REST de manera intencionada, mientras que otros las cumplen de manera accidental).

5. Conclusiones

El análisis realizado nos permite ver las siguientes carencias en la arquitectura:

- Hoy por hoy, el monopolio de los servicios Web es mantenido por SOAP.

- Existe una carencia de documentación REST bastante importante (sufrida al tratar de encontrar documentación para éste artículo, pues las fuentes son bastante reducidas).
- Existen escasas implementaciones de referencia y ejemplos prácticos que acerquen la arquitectura al programador común.

La solución aparente pasa por la creación de una organización o entidad que agrupe el disperso y escaso trabajo existente sobre REST.

6. Referencias

6.1. Artículos

[1] Fielding, Roy T. "Architectural Styles and the Design of Network-based Software Architectures." Tesis Doctoral, Universidad de California, 2000.
http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[2] Przybilski, Michael. "REST: Representational State Transfer". Universidad de Helsinki, 2006
www.cs.helsinki.fi/u/chande/courses/cs/MWS/reports/MichaelPrzybilski_REST.pdf

[3] Prescod, Paul. "REST and the Real World"
http://www.xml.com/pub/a/2002/02/20/rest.html

[4] Prescod, Paul. "SOAP, REST and Interoperability"
http://www.prescod.net/rest/standardization.html

[5] CuboVelázquez, Alberto. "Representational State Transfer (REST). Un estilo de arquitectura para Servicios Web. Panorámica y estado del arte."
http://trajano.us.es/~antonio/proyecto-REST.doc#_Toc139162493

[6] Tim Berners-Lee. The world wide web: Past, present and future.
http://www.w3.org/People/Berners-Lee/1996/ppf.html.

6.2. Wikipedia

[7] http://es.wikipedia.org/wiki/URI

[8] http://es.wikipedia.org/wiki/Servicio_Web

[9] http://en.wikipedia.org/wiki/Representational_State_Transfer

[10] http://en.wikipedia.org/wiki/SOAP

6.3. Recursos Web:

[11] Consorcio W3C. http://www.w3c.org

[12] OpenSearch: http://opensearch.a9.com/

[13] Amazon Developers site: http://www.amazon.com/gp/aws/landing.html

[14] Ebay developer site: http://developer.ebay.com/rest/

[15] Yahoo developer site http://developer.yahoo.com/

[16] YouTube developer site http://www.youtube.com/dev_rest

[17]Flickr developer site http://www.flickr.com/services/api/

6.4. Especificaciones

[RFC 1738], URL: http://www.faqs.org/rfcs/rfc1738.html

[RFC 2396], URI: http://www.faqs.org/rfcs/rfc2396.html

[RFC 1945], HTTP: http://www.faqs.org/rfcs/rfc1945.html

[W3C-SOAP], SOAP:http://www.w3.org/TR/2002/WD-soap12-part0-20020626/