

Capítulo 1

Arquitectura

1.1. ¿Qué es REST?

REST¹ es un estilo arquitectural para sistemas distribuidos como la web. El termino se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP². REST define una serie de normas o principios para el diseño de arquitecturas en red. Estas normas establecen como son definidos y diseccionados los recursos del sistema a desarrollar.

1.2. ¿Cuál es la motivación de REST?

La principal motivación de REST es capturar las características de la Web que la han hecho tan exitosa a la hora de diseñar un servicio web. Se puede ver que la Web ha sido la única aplicación distribuida que ha conseguido ser escalable al tamaño de Internet. Parte de su éxito se debe a su esquema de direccionamiento global (estándar y extensible a la vez). Dentro de la Web se define el espacio de URIs. Una URI³ representa un recurso, el cuál es un objeto conceptual. El tipo de recurso identificado puede ser un servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.

REST afirma que la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST).

¹REST: Representational State Transfer

²HTTP: HiperText Transfer Protocol

³URI: Uniform Resource Identifier

- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en si define un conjunto pequeño de operaciones, las más importantes son GET, POST, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD⁴ que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.
- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.
- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

1.3. ¿Cuáles son los principios de REST?

El estilo arquitectural que subyace en la Web es el modelo REST. Los objetivos de este estilo son:

- La escalabilidad en la interacción con la gran variedad de componentes como son en la actualidad: estaciones de trabajo, sistemas industriales, dispositivos móviles, etc. El crecimiento de la Web ha sido de tipo exponencial sin degradar su rendimiento.
- La generalidad de las interfaces. El protocolo HTTP permite la interacción de cualquier cliente con un cualquier servidor HTTP sin una configuración específica. Esto no es del todo cierto en otras alternativas de diseño como puede ser SOAP para servicios Web.
- Puesta en funcionamiento independiente. Los servidores antiguos deben ser capaces de entenderse con clientes actuales y viceversa. HTTP permite la extensibilidad mediante el uso de cabeceras, a través de las URIs, etc.
- La compatibilidad con componentes intermedios. Las cachés se usan para mejorar el rendimiento, los firewalls permiten mejorar las políticas de seguridad, los gateways permiten encapsular sistemas no Web y los proxies nos permiten una gestión y escalabilidad de las comunicaciones. En resumen, los componentes intermedios nos ayudan a reducir la latencia de las comunicaciones, reforzar la seguridad y encapsular otros sistemas.

REST cumple dichos objetivos aplicando los siguientes criterios:

- Identificación y manipulación de recursos por medio de representaciones (URIs). Los recursos son los objetos lógicos a los que se les envían los mensajes y no pueden ser directamente accedidos ni modificados. Más bien se trabaja con una representación

⁴CRUD: Create, Retrieve, Update y Delete

de ellos. La representación interna del recurso puede ser de cualquier tipo como por ejemplo una base de datos relacional o simplemente un fichero de texto.

- Uso de mensajes autodescriptivos. REST dicta que los mensajes HTTP deberían ser lo más descriptivos posible. Esto ayuda a que los intermediarios interpreten mejor el servicio y puedan realizar tareas sobre él de forma directa y autónoma. HTTP logra ésto último por medio de cabeceras en los mensajes, uso de métodos estándares y un mecanismo de direccionamiento global. Un ejemplo sería el uso del método GET, el cuál las cachés saben que este método es cacheable mientras que el método POST no lo es por su significado natural. Además HTTP proporciona a los intermediarios un método para conocer información del estado de un recurso (HEAD) como puede ser la caducidad de la información que suministra el propio recurso.
- Hipermedia como mecanismo de estado de la aplicación. El estado de la aplicación debería ser capturada en uno o más documentos de hipertexto.

1.4. ¿Como sería un diseño basado en REST?

Las acciones CRUD fueron diseñadas para operar con datos atómicos dentro de una transacción de base de datos. Un diseño REST puede ser visto como la transferencia de un documento de una aplicación a otra en un estado más complejo.

Significado de las operaciones en un diseño REST:

Acción	HTTP	SQL
Create	PUT	INSERT
Read	GET	SELECT
Update	POST	UPDATE
Delete	DELETE	DELETE

Cuando utilizamos REST, no se utiliza ningún tipo de mecanismo externo a HTTP para establecer un estado entre el cliente y el servidor, como pueden ser las cookies. El uso de este tipo de técnicas pone en riesgo la privacidad y la seguridad de las transacciones en un modelo cliente/servidor.

1.5. REST vs SOAP

Dada la cierta complejidad de diseñar un Servicio Web basado en SOAP, muchos diseñadores de grandes empresas como Google, eBay, Yahoo! o Amazon están empezando a diseñar servicios basados en REST cuando se trata de manejar una cantidad masiva de información.

El problema principal radica del propósito inicial de SOAP. Esta tecnología fué originalmente pensada para ser una versión sobre Internet, de DCOM⁵ o CORBA⁶. Estas

⁵DCOM: Distributed Component Object Model (Modelo de Objetos de Componentes Distribuidos)

⁶CORBA: Common Object Request Broker Architecture (Arquitectura Común de Intermediarios en Peticiones de Objetos)

tecnologías lograron un éxito limitado debido a que estaban basadas en modelos RPC⁷. Este tipo de modelos son más adecuados en entornos aislados y perfectamente conocidos. Cuando el número de usuarios es muy grande es necesario usar una estrategia de diseño distinta.

1.6. ¿Como diseñar un servicio Web basado en REST?

Lo que primero se debe de hacer a la hora de diseñar un servicio Web RESTful es identificar los recursos. Los recursos del servicio se identifican mediante URIs. Hay que prestar especial atención a este punto ya que si sólo se define un punto de acceso nos estaremos basando en RPC y no en un diseño REST. Para ayudarnos a la hora de la localización de los recursos del sistema debemos prestar especial atención en las colecciones y en las interfaces de búsqueda. Una colección de recursos en sí también representa un recurso. Una interfaz de búsqueda es un recurso, ya que el resultado de la búsqueda es otro conjunto de recursos. Un punto importante en este apartado es llevar a cabo una definición de URLs para los recursos que sean nombres y no verbos (acciones), ya que las acciones sobre los recursos serán definidos con posterioridad usando los métodos HTTP estándares. Un ejemplo de esto último es el siguiente:

<i>No REST: <code>http://www.service.com/profiles/getProfile?id=001</code></i> <i>REST: <code>http://www.service.com/profiles/001</code></i>

Una vez definidos los recursos se tiene que decidir cuál va a ser su representación, es decir, el formato con el que podrá ser accedido por las aplicaciones externas al sistema. Un mismo recurso lógico puede ser representado por más de un formato con lo que dará lugar a varios recursos físicos. Un recurso puede ser representado a través de un documento HTML, XML, JSON, una imagen, etc. A la hora de decidir el formato de representación de los recursos es recomendable hacer uso de formatos estándares para facilitar que nuestro sistema se componga con otros externos. El formato de representación más extendido para el intercambio de información es XML. Un ejemplo de este tipo de formato es el siguiente:

<pre><Jugador xmlns='http://example.org/my-example-ns/'> <nombre>Nombre completo aqui.</nombre> <telefono>Número de teléfono aqui.</telefono> </Jugador></pre>
--

Una vez definidos los recursos y su representación tenemos que definir como van a ser referenciados los recursos, es decir, los métodos que va a soportar cada recurso. El acceso a los recursos es a través de la URI correspondiente y de los métodos que soporte el recurso lógico definido. El acceso puede ser de diferentes formas, ya que se puede recibir una representación del recurso (GET o HEAD), se puede modificar o crear un recurso (POST o PUT) o se puede eliminar un recurso (DELETE).

⁷RPC: Remote Procedure Call (Llamada a Procedimiento Remoto).

HTTP	CRUD	Descripción
PUT	Create	Crear un nuevo recurso
GET	Read	Obtener la representación de un recurso
POST	Update	Actualizar un recurso
DELETE	Delete	Eliminar un recurso

Además de conocer el tipo de representación que se puede recibir de un recurso, es importante también conocer los códigos de estado HTTP (200, 301, 400, 410...) y sus significados que pueden ser devueltos tras la transacción.

Ninguna representación debería quedar aislada. Se recomienda poner hipervínculos dentro de la representación de un recurso para permitir de esta forma que los clientes puedan obtener más información acerca del mismo.

También sería recomendable describir como ha de ser invocado nuestro servicio mediante un documento WSDL/WADL o simplemente un HTML⁸.

1.7. ¿Cuáles son las características de REST y SOAP?

	REST	SOAP
Características	Las operaciones se definen en los mensajes. Se define cada recurso con una única dirección. Cada objeto soporta los métodos estándares HTTP. Componentes débilmente acoplados.	Las operaciones son definidas como puertos WSDL. Dirección única para todas las operaciones. Múltiples instancias del proceso comparten la misma operación. Componentes fuertemente acoplados.
Ventajas	Bajo consumo de recursos. Las instancias del recurso son creadas explícitamente. El cliente no necesita información de enrutamiento a partir de la URI. Los clientes pueden usar una interfaz tipo 'listener' para notificaciones. Generalmente fácil de construir y de adoptar.	Fácil de usar (generalmente). La depuración es más fácil. Las operaciones complejas pueden ser escondidas tras una fachada. Envolver APIs existentes es sencillo. Incrementa la privacidad.
Desventajas	Gran número de objetos. Descripción sictáctica/semántica informal (orientada al usuario).	Los clientes necesitan conocer las operaciones y su semántica antes de su uso. Las instancias de los objetos son creadas implícitamente.

⁸HTML: HyperText Markup Language (Lenguaje de Etiquetas de Hipertexto)

Bibliografía

- [1] Leonard Richardson & Sam Ruby, “ RESTfull Web Services”. O’REILLY, May 2007
- [2] Hahn, J. “ \LaTeX for eveyone”. Prentice Hall, New Jersey, 1993.

VERSION 0.1