Servicios Web

Franco Bocalon, Luis Guanuco, Santiago Nolasco
Sistemas Distribuidos
Especialidad en Sistemas Embebidos
Instituto Universitario Aeronáutico

***Resumen*—Las arquitecturas de los servicios web fueron evolucionando en función de las necesidades tanto de los proveedores como los usuarios. La expansión de internet y nuevas tecnologías (dispositivos móviles, IoT, etc.) modelaron los sistemas actuales. En base a la definición de SOA (Services Object Architecture), se enfocará en la importancia y madurez de los servicios SOAP (Simple Object Access Protocol) para luego**

Aquí se expone un resumido análisis del *Estado del Arte* de los servicios web con el foco en los *Servicios REST*.

***Palabras claves*—Web services, SOA, SOAP, REST.**

I. Introducción

En la actualidad, debido a los competitivos mercados globales, las compañías se ven presionadas a responder de la manera más efectiva posible. Saber actuar ante los cambios que afectan de manera natural a los negocios, optimizar los procesos, reducir los costos de IT, y lograr flexibilidad, son algunos de los factores claves para la competitividad y el crecimiento de las organizaciones. Para lograrlo se necesita una herramienta basada en estándares para integrar sistemas y aplicaciones heterogéneos sobre una serie de plataformas y protocolos de comunicación, con una metodología bien establecida. Este marco de trabajo conceptual es SOA (*Service-Oriented Architecture*).

A. SOA

La Arquitectura Orientada a Servicios supone una estrategia general de organización de los elementos de IT, de forma que una colección desordenada de sistemas distribuidos y aplicaciones complejas se pueda transformar en una red de recursos integrados, simplificados y sumamente flexible[4]. Un proyecto SOA bien ejecutado permite alinear los recursos de IT de forma más directa con los objetivos de negocio, ganando así un mayor grado de integración con clientes y proveedores, esto es descomponiendo toda su estructura en servicios. Los cuales pueden ser añadidos y también sustraídos sin alterar el complejo sistema. Esta independencia abstrae al programador de las conexiones entre plataformas, lenguajes de programación y dependencia de otros servicios.

Los resultados son la reducción de costos de implementación, innovación de servicios a clientes, adaptación ágil ante cambios y reacción temprana ante la competitividad, ya que, combinan fácilmente las nuevas tecnologías con aplicaciones independientes permitiendo que los componentes del proceso se integren y coordinen de manera efectiva y rápida.

Tabla I
Arquitecturas de servicios web.

Arquitectura tradicional	Arquitectura orientada a servicios
Forzada a la funcionalidad	Orientado al proceso
Diseñado para durar	Diseñado para el cambio
Desarrollo con largos ciclos de tiempo	Desarrollo iterativo
Fuerte acoplamiento	Bajo acoplamiento
Aplicación especifica	Heterogéneo
Orientado a datos	Orientado al servicio empresarial

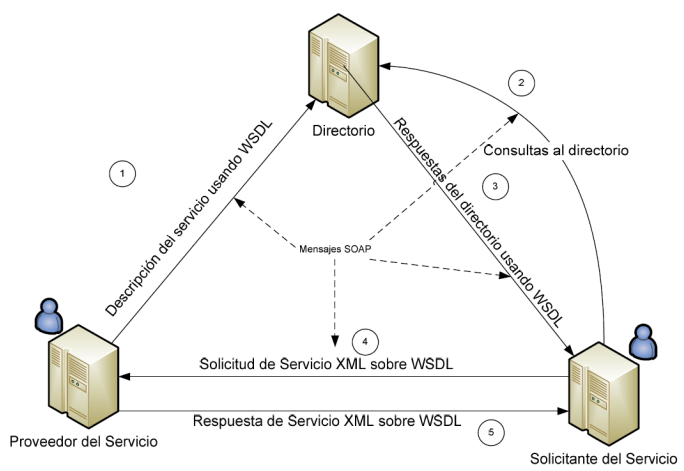

Fig. 1. Interacción de mensajes en las arquitecturas SOA.

B. SOA vs Arquitectura tradicional

En la Tabla I podemos apreciar la diferencia entre una arquitectura tradicional y el paradigma SOA[5]. La diferencia sustancial es que en la arquitectura SOA, se busca la independencia de los servicios. Esta independencia se logra a traves de la aplicacion de protocolos, logrando un bajo acoplamiento entre los servicios web.

II. Servicios basados en SOA

El uso más común que se suele dar a este servicio es el de integrar diferentes sistemas o componentes de una o varias plataformas. Mediante el uso de estándares se consigue dicha integración y a la vez interoperante. La clave se encuentra en el intercambio de mensajes entre sistemas (Figura 1)[6].

Los servicios web se basan en un conjunto de estándares de comunicación. A continuación se describirán algunos de estos estándares.

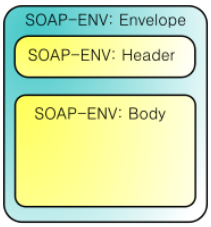

Fig. 2. Estructura del mensaje utilizado por SOAP.

A. XML

Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos[1] para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información.

B. WSDL

Describe la interfaz pública a los servicios web. Está basado en XML y describe la forma de comunicación. Es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

WSDL se usa a menudo en combinación con SOAP y XML Schema[REF]. Un programa cliente que se conecta a un servicio web puede leer el WSDL para determinar qué funciones están disponibles en el servidor. Los tipos de datos especiales se incluyen en el archivo WSDL en forma de XML Schema. El cliente puede usar SOAP para hacer la llamada a una de las funciones listadas en el WSDL.

C. SOAP

SOAP (*Simple Object Access Protocol*) es usado para el intercambio de datos. Es un paradigma de mensajería de una dirección sin estado que puede ser utilizado para formar protocolos más complejos y completos según las necesidades de las aplicaciones que lo implementan. Puede formar y construir la capa base de una *pila de protocolos de web service*, ofreciendo un framework de mensajería básica en el cual los *web services* pueden ser construidos.

Este protocolo está basado en XML y la estructura consta de tres partes[REF]:

- *Envelope*: Qué hay en el mensaje y cómo procesarlo.
- *Header*: Es una extensión que permite enviar información relativa a cómo debe ser procesado el mensaje.
- *Body*: Contiene la información relativa a la llamada y la respuesta.

El protocolo SOAP tiene tres características principales:

- *Extensibilidad*: seguridad y WS-routing son extensiones aplicadas en el desarrollo.
- *Neutralidad*: SOAP puede ser utilizado sobre cualquier protocolo de transporte como HTTP, SMTP, TCP o JMS.
- *Independencia*: SOAP permite cualquier modelo de programación.

El enfoque del SOAP[7] es encapsular la lógica como solicitud de la base de datos para el servicio. A traves de un método (o función) en cualquier lenguaje como C, VB, Java, etc.; y a continuación establecer un proceso que escucha las solicitudes al servicio siendo dichas solicitudes en formato SOAP y que contienen el nombre del servicio y los parámetros requeridos. La capa de transporte puede ser HTTP, aunque podría ser fácilmente SMTP o alguna otra. Entonces, el proceso de escucha, que por simplicidad se suelen escrito en el mismo lenguaje que el método de servicio, decodifica la petición SOAP de entrada y la transforma en una invocación del método. Luego toma el resultado de la llamada al método, lo codifica en un mensaje SOAP (respuesta) y lo envía de vuelta al solicitante. Conceptualmente, esta disposición tiene el aspecto de la Figura 3.

Existen más especificaciones, a las que se denomina genéricamente como la arquitectura WS-*, que definen distintas funcionalidades para el descubrimiento de servicios Web, gestión de eventos, archivos adjuntos, seguridad, gestión y fiabilidad en el intercambio de mensajes y transacciones.

III. REST

REST o *Representational State Teansfer* fue creado por Roy Fielding[2] en el año 2000. Este concepto fue expuesto por primera vez en la disertación de la tesis doctoral de Fielding. En dicha exposición se exploraba varios estilos arquitectónicos basados en redes. En el Capítulo 5 [1] de sus tesis se hace una descripción profunda. REST puede ser descrito como una forma de construir servicios Web siguiendo alguna restricciones entre el proveedor y el cliente de tales servicios como son[2]:

- *Mantener separados los intereses entre el cliente y servidor*: con esto, los clientes no deberían preocuparse por el almacenamiento de datos, al igual que los servidores no deben preocuparse por las interfaces de usuario. Esto desacopla por completo el desarrollo de ambas partes del sistema y permite economizar los recursos en el escalamiento.
- *La comunicación entre cliente y servidor debería ser "sin estados" (stateless)*: los servidores no deberían almacenar ninguna información sobre el contexto del cliente en sus llamadas. Obviamente con las excepciones de sesiones que utilizan información para mantener la autenticación de la conexión.
- *Los clientes debe almacenar en cache las respuestas*: todas las respuestas del servidor deben conener la suficiente información relacionada a cache. Por lo tanto, los clientes puede confiar en esa información y decidir por sí mismos qué almacenar de la respuesta en cache.

[1]De la misma manera que HTML es a su vez un lenguaje definido por SGML.

[2]`http://www.ics.uci.edu/~fielding/`

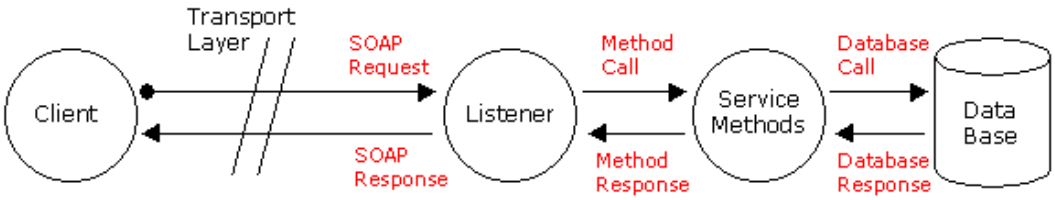

Fig. 3. Componentes en un pedido de servicios SOAP.

- *La conexiones pueden producirse a través de varias capas de comunicación*: los clientes no deben ser capaces de distinguir sí están directamente conectados al servidor de la aplicación o a un agente intermediario que transmite la información.

La simplicidad de REST puede verse en el siguiente ejemplo. Se pide obtener los datos de un usuario a partir de su identificador. Usando la arquitectura SOAP, la petición debería tener la siguiente forma:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

Esto se envía al servidor utilizando el comando `HTTP POST`. El resultado probablemente será un archivo XML, pero esto se encuentra embebido como la carga-paga, dentro de una respuesta SOAP.

Y con REST la consulta probablemente sea como la que sigue,

```
http://www.acme.com/phonebook/UserDetails/12345
```

En este caso solo se tiene un URL. Este URL se envía al servidor usando simplemente un pedido de `GET` y la respuesta en HTTP se obtiene en bruto. No se adquiere nada más de lo que se pide al servidor.

Note como el método del URL no se llama `GetUserDetails`, es simplemente `UserDetails`. Esto es una de las convenciones en el diseño REST donde se usa *sustantivos* en vez de *verbos* para denotar un simple *recurso*.

Una analogía que resume las diferencias entre SOAP y REST es considerar ambos servicios como el redactar una carta. Donde el uso de SOAP sería como utilizar *un sobre para enviar la carta*. En cambio REST podría considerarse una *postal*. Una postal es fácil de enviar y leerla, además de consumir menos papel.

Con respecto a la seguridad se podría decir que REST es un poco más seguro que SOAP. En particular, REST podría ser implementado sobre un socket seguro como ser HTTPS y el contenido podría también estar encriptado usando cualquier mecanismo existente. De todas formas, sin encriptación, tanto REST como SOAP son inseguros.

IV. Comparación de servicios web

Aquí vamos a poner las ventajas y desventajas entre los servicios REST y los web services tradicionales.

SOAP proporciona formas de acceder y manipular objetos remotos, mientras que REST se focaliza en la operaciones que puede ser ejecutada sobre los recursos. Esta característica hace que REST tenga mayor adopción en APIs. Además, porque REST hereda operaciones del protocolo HTTP. Esto último hace fácil la elección de "cómo abrir una API web". Estas características han influido tanto que grandes compañías Web utilizan la API REST[3].

REST es mejor que SOAP en situaciones que no requieren un mapeo total de los objetos al cliente. La comunicación de información de objetos en ambos sentidos demanda una cantidad importante de ancho de banda, lo que podría ser una limitante en entornos donde la conectividad es costosa. Una API requerida principalmente para aplicaciones móviles es uno de los casos de uso donde se debe evitar el uso de SOAP.

Otro factor en la preferencia a REST es la simplicidad de la implementación con respecto a SOAP. También, al ser menos restrictivo que SOAP, algunas APIs de REST trabajan mejor en la actualidad. En algunos casos los desarrolladores no conocen en profundidad los contratos a la hora de implementar una API. Las APIs públicas cambian constantemente debido a las necesidades de negocio y son adaptables rápidamente. En esta situación REST resulta una elección natural. El uso de SOAP en las condiciones expuestas sería contraproducente ya que podría introducir una complejidad contractual innecesaria.

Actualmente es común leer el término *ROA* (REST Oriented Architecture). ROA es simplemente el nombre que se le da a los servicios SOA (Service Based Architecture) que utilizan REST.

La principal ventaja de SOA sobre ROA es la madurez de las herramientas. Sin embargo, esto ha cambiado con el tiempo. En la actualidad la mayoría de los servicios web se basan en REST.

Como se dijo anteriormente, la principal ventaja de REST es la facilidad en su implementación, agilidad en el diseño y el enfoque sencillo en el manejo de recursos. En cambio, SOAP tiene un perfil orientado a servicios para empresas. Aquí se pueden encontrar usos en bancos e industrias financieras. Contrariamente, si alguien necesita montar rápidamente un servicio con un buen funcionamiento y baja demanda de recursos, seguramente elegirán REST.

[3]RESTful es el nombre técnico de la implementación de REST.

V. Conclusiones

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Referencias

[1] Roy Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine. 2000. url:http://www.ics.uci.edu/\texttildelowfielding/pubs/dissertation/rest_arch_style.htm.

[2] Bruno Pedro, *Is REST better than SOAP? Yes, in Some Use Cases*, url: http://nordicapis.com/rest-better-than-soap-yes-use-cases/.

[3] Dr. M. Elkstein, *Learn REST: A Tutorial*, url: http://rest.elkstein.org/.

[4] Whitepaper, *La Arquitectura SOA de Microsoft Aplicada al Mundo Real*.

[5] Umair Ashraf, *CS 415 N-Tier Application Development*. National University of Computer and Emerging Sciences. July 5, 2013.

[6] Miguel Rodríguez, *SOA vs SOAP y REST*. url: http://www.adictosaltrabajo.com/tutoriales/soavs-soap-rest/.

[7] Nicholas Quaine, *SOAP Basics – What is SOAP?* . url: http://www.soapuser.com/basics1.html.