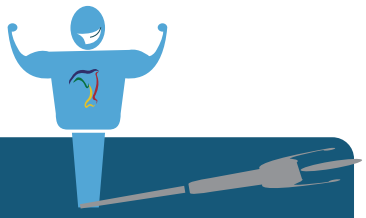
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

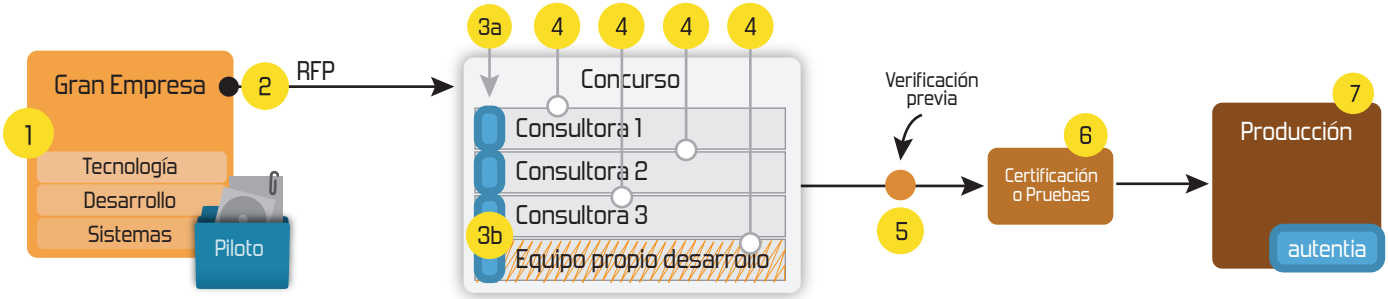
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Miguel Arlandy Rodríguez

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2014-01-10

SOA vs. SOAP y REST.

0. Índice de contenidos.

1. Introducción

SOA y SOAP suelen ser términos que, a menudo, suelen generar bastante confusión (supongo que por la semejanza de sus siglas). Parece que muchas veces no tenemos del todo claro dónde empieza y acaba cada cosa aunque, en el fondo, sabemos que existe relación entre ambos términos. Incluso es común encontrar por ahí algunos artículos donde se refieren a SOA cuando realmente quieren decir SOAP.

Con el desarrollo de servicios web basados en REST también puede ocurrir algo parecido.

En este tutorial intentaremos explicar qué diferencias existen entre una arquitectura SOA y el desarrollo de servicios web basados en SOAP o REST, cómo se relacionan y los beneficios esperados de cada uno.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

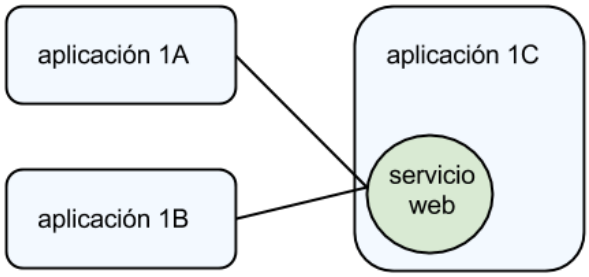
- Hardware: Portátil MacBook Pro 15' (2.2 Ghz Intel Core I7, 8GB DDR3).
- Sistema Operativo: Mac OS X Mavericks 10.9

3. ¿Por qué servicios web basados en SOAP o REST?.

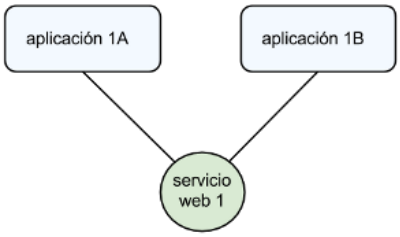
Nótese que este punto no pretende ser una comparativa entre SOAP y REST, lo que intentaremos es destacar los objetivos y características comunes del desarrollo de servicios web basados en SOAP o REST.

SOAP y REST son siglas asociadas a estándares para el diseño y desarrollo de web services o servicios RESTful. El uso más común que se suele dar a estos servicios es el de **integrar** diferentes sistemas o componentes de una o varias plataformas. Mediante el uso de estándares conseguimos que esa integración se convierta en **interoperabilidad**. La interacción se consigue mediante el intercambio de mensajes entre sistemas.

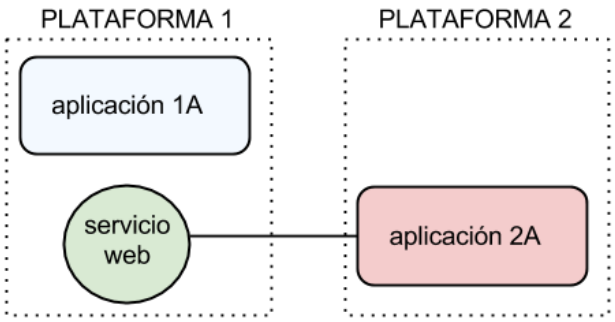
Es muy común ver cómo se emplean tanto SOAP como REST para **exponer parte de la funcionalidad** (o recursos) de diferentes aplicativos. Tenemos ejemplos muy claros de esto en dos excelentes herramientas como son Sonar y Jenkins. En este caso exponen recursos a través de un API REST (imagino que por sencillez y porque los requisitos lo permiten) para que otros sistemas puedan interoperar con ellos. Lógicamente, esta misma filosofía es aplicable dentro de una plataforma empresarial.



También puede ser bastante común crear un web service cuya lógica funcionalidad sea requerida por distintos aplicativos o componentes dentro de una plataforma (el típico servicio de utilidad). Este enfoque ya se va acercando a SOA, pero eso será en el siguiente punto. Además, podríamos tener un servicio de más alto nivel para ser consumido desde diferentes frontales: ej. una aplicación web y una aplicación móvil.



Y un ejemplo clásico en el uso de servicios web es el de **integrar** diferentes sistemas o plataformas como puede ser el caso de diferentes departamentos dentro de una organización o diferentes organizaciones.



Además de la interoperabilidad, otra gran ventaja que ofrecen estas alternativas es la **flexibilidad a la hora de elegir la tecnología** con la que queremos implementar la lógica de negocio que queremos exponer. De esta forma, podemos comunicar diferentes sistemas o componentes independientemente de la tecnología con la que están implementados.

Pero, ¿qué tiene que ver todo esto con SOA?. Lo vemos a continuación...

4. ¿Por qué SOA?.

SOA es un modelo de arquitectura tecnológica que surge de la aplicación del paradigma de orientación a servicios. Dicho paradigma no es una idea revolucionaria, sino que **surge de la influencia de diferentes modelos** como pueden ser: la orientación a objetos, BPM, orientación a aspectos, web services...

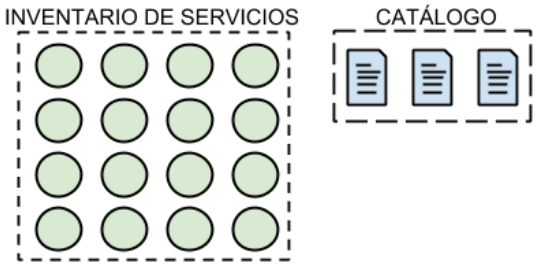
La idea subyacente consiste en descomponer la lógica de negocio de una organización (o partes de ella) en pequeñas unidades de funcionalidad. Estas pequeñas unidades son los **servicios**. Con esto conseguimos romper con el concepto de aplicaciones "silo", donde se creaba una aplicación para resolver una necesidad de negocio concreta, otra para resolver otra, etc... Lo que tendremos será una **plataforma transversal** formada por un **inventario de servicios** (o varios) de forma que no solventaremos las necesidades cambiantes del negocio creando nuevas aplicaciones sino **combinando diferentes servicios** (y creando nuevos servicios cuando corresponda). De esta forma conseguimos que los departamentos de IT y negocio estén alineados de forma que el primero pueda responder de manera ágil a las exigencias del segundo.

Dicho esto, alguien puede pensar: *"Entonces para tener una SOA lo que tengo que hacer es crear web services a lo bestia"*. Pues no, ójala fuese tan fácil :).

Es cierto que para poder tener una arquitectura SOA necesitamos tener una buena base de servicios. Pero no vale con crearlos de cualquier manera. Estas son algunas consideraciones que debemos tener en cuenta:

- Debemos contar con una buena base de servicios **reutilizables** o muli-propósito. Servicios no diseñados para resolver una necesidad de negocio específica. Los servicios de utilidad y entidad son perfectos en este sentido. De esta forma podremos crear múltiples **composiones** basadas en ellos conforme las necesidades de negocio van cambiando.
- Nuestros servicios deben contar con un contrato estandarizado (ya sea WSDL, un documento o ambos) con un modelo de datos normalizado dentro de todo nuestro inventario. De esta manera facilitamos la interacción entre servicios. Sustituimos el concepto de integración por **interoperabilidad intrínseca**.
- Debemos **categorizar y registrar** todos nuestros servicios (recursos de plataforma) de forma que, en cualquier momento, podamos consultar con qué recursos contamos, qué funcionalidades encapsulan y cómo interacturar con ellos.
- Debemos evitar el solapamiento de funcionalidades entre servicios como parte de la labor de **gobierno SOA** que debemos ejercer sobre nuestra plataforma.
- Debemos evitar "casarnos" con una tecnología en concreto (Java, .Net, etc...) a la hora de crear nuestros

servicios. Los servicios deben poder implementarse con la tecnología que consideremos necesaria en cada momento sin que esto afecte al resto de la plataforma.



Como vemos, no resulta tan fácil diseñar una arquitectura orientada a servicios. Sin embargo, los servicios web bastados tanto en SOAP como REST son **excelentes opciones** a la hora de crear los servicios que conformarán nuestra plataforma ya que, sin lugar a dudas, reunen las condiciones ideales para poder diseñar servicios que cumplan con los principios de diseño alineados con SOA.

5. La relación entre SOA y los servicios web basados en SOAP o REST.

Como comentamos anteriormente, SOA gira en torno a los servicios o recursos de plataforma. Dichos servicios deben cumplir con una serie de **principios de diseño** como pueden ser: una alta capacidad de reutilización, abstracción, bajo acoplamiento, autonomía, capacidad de composición, contar con un contrato estandarizado (WSDL en caso de SOAP, convenciones en caso de REST y posiblidad de complementarlos con documentación en ambos casos)...

Tanto SOAP como REST pueden cumplir a la perfección con cualquiera de estos principios. Sin embargo, son especificaciones totalmente independientes a SOA. De la correcta aplicación del paradigma de orientación a servicios y, sobre todo, del cumplimiento de los **principios de diseño de servicios** surgirá el éxito o el fracaso de esta relación.

Lógicamente, **no vale únicamente con tener servicios web para tener una SOA**. Como vimos en el punto anterior, los servicios que exponen Jenkins y Sonar son servicios de muy alto nivel (servicios de tarea). Si todo nuestro inventario se basase en este tipo de servicios, no podríamos realizar composiciones complejas ni, por tanto, modelar de forma ágil diferentes procesos cuando el negocio así lo requiriese.

Un fallo conceptual muy común en algunos equipos de IT es el de asegurar que tienen una fabulosa arquitectura SOA cuando realmente lo que tienen son unos cuantos servicios de alto nivel orquestados por un bus. Pero eso no es SOA, en todo caso es integración.

6. Referencias.

- SOA: Principles of Service Design

7. Conclusiones.

Construir una arquitectura SOA es un trabajo relativamente complejo, laborioso y siempre a medio/largo plazo. No todas las plataformas u organizaciones son las mejores candidatas para ello. Es muy importante conocer bien el paradigma de orientación a servicios antes de barajar esta posibilidad. Debemos conocer antes sus ventajas, inconvenientes, objetivos, beneficios, cambios a nivel organizativo, etc...

Podemos diseñar excelentes arquitecturas basadas en REST de una manera bastante más sencilla. Incluso podemos basarnos en aplicaciones silo, que tienen sus inconvenientes pero también sus ventajas, para la integración entre ellas (si fuese necesaria) ya tenemos muchas alternativas como los servicios web. No todo debe pasar por SOA. SOA simplemente es un medio para obtener una serie de beneficios, nunca debe ser un fin en sí mismo. Todo depende del tamaño de la organización, presupuesto, lo cambiante del negocio y, sobre todo, del sentido común :-).

Espero que este tutorial os haya sido de ayuda. Un saludo.

Miguel Arlandy

marlandy@autentia.com

Twitter: @m_arlandy

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share |

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**