ARQUITECTURA DE SOFTWARE ORIENTADA A SERVICIOS

Autor: Luciano Straccia

Versión 2022.02

Temario

- Desarrollo de software basado en componentes
- Servicios Web
- Microservicios
- Comunicación entre servicios y Servicios REST

Desarrollo de software basado en componentes (CBSE)

Desarrollo de software basado en componentes

- El desarrollo de software basado en componentes (CBSE, Component Based Software Engineering) es aquel que está fundamentado en la producción de diversas piezas de software ensambladas de una manera integral que permita el funcionamiento del sistema software como un todo.
- En este contexto, se define a un sistema como “un conjunto de mecanismos y herramientas que permiten la creación e interconexión de componentes software, junto con una colección de servicios para facilitar las labores de los componentes que residen y se ejecutan en él”.

Componentes

- Un componente es una pieza de código preelaborado que encapsula alguna funcionalidad expuesta a través de interfaces estándar.

- Componentes: “unidad de composición de aplicaciones software que posee un conjunto de requisitos y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto por otros componentes, de forma independiente en tiempo y forma”.

Ventajas

- Reutilización de software
- Simplificación de las pruebas
- Simplificación del mantenimiento del sistema
- Una mayor calidad de los componentes.

Servicios Web

Servicios Web

- Un servicio puede ser definido como un componente de software reutilizable de acoplamiento flexible que encapsula la funcionalidad discreta que puede ser distribuido y accesible mediante programación.
- Un servicio web es un servicio que se accede a través de protocolos de Internet y basado en XML estándar
- Una distinción fundamental entre un servicio y un componente tal como se define en CBSE es que los servicios son independientes
- Se deben definir: las operaciones compatibles con el servicio, el formato de los mensajes enviados y recibidos por el servicio y dónde se encuentra y cómo se accede al servicio

Ejemplo Servicio Web

- http://www.afip.gob.ar/ws/
- Web Service de Factura Electrónica ("wsmtxca"), comprobantes "A" y "B" con detalle de items, CAE y CAEA.

Microservicios

Microservicios

- Caracterización particular de los Servicios Web
- El concepto de arquitectura microservicio ha surgido en los últimos años para describir una forma particular de diseñar aplicaciones de software como suites de servicios con independencia de despliegue. Si bien no existe una definición precisa de este estilo arquitectónico, hay ciertas características comunes alrededor de organización en torno a la capacidad del negocio, el despliegue automatizado, la inteligencia en los puntos finales, y el control descentralizado de los lenguajes y los datos.
- La arquitectura de microservicios mantiene un sistema similar a un gobierno descentralizado, donde cada módulo contará por ejemplo con su propia base de datos, en lugar de acudir todos a la misma sobrecargándola así de solicitudes y arriesgándonos a que si falla ésta, todas la aplicación caiga.

Comunicación entre servicios

Stateful y Stateless

- El **estado** es la información que los servidores mantienen acerca de la interacción que se lleva a cabo con los clientes.
- La comunicación puede establecerse con estado (stateful) o sin estado (stateless)
- Stateful generalmente hace más eficiente el comportamiento de los servidores con información. Información muy breve mantenida en el servidor puede hacer más chicos los mensajes o permite producir respuestas más rápido.
- Pero también stateful es fuente de errores: mensajes del cliente pueden perderse, duplicarse llegar en desorden. El cliente puede caerse y rebootear, con lo cual la información que tiene el servidor de él es errónea y también sus respuestas

Stateless

- Sin estado
- Se trata cada petición como una transacción independiente que no tiene relación con cualquier solicitud anterior, de modo que la comunicación se compone de pares independientes de solicitud y respuesta
- Un protocolo sin estado no requiere que el servidor retenga información de la sesión o de estado acerca de cada socio de las comunicaciones durante la duración de múltiples peticiones
- Stateless Protocol: IP – HTTP

Stateless

- El diseño sin estado simplifica el diseño del servidor porque no hay necesidad de asignar dinámicamente almacenamiento para tratar las conversaciones en curso. Si un cliente desaparece en medio de la transacción, ninguna parte del sistema tiene que ser responsable de limpiar el estado actual del servidor.
- Una desventaja de los protocolos sin estado es que puede ser necesario incluir información adicional en cada petición, y esta información adicional necesitará ser interpretada por el servidor.

Stateful

- Con estado
- El servidor debe retener información de la sesión o de estado acerca de cada socio de las comunicaciones durante la duración de múltiples peticiones
- Stateful Protocol: FTP
- Un protocolo FTP lleva a cabo una sesión interactiva con el usuario. Durante la sesión, al usuario se le proporciona un medio para autenticar y establecer diversas variables (directorio de trabajo, modo de transferencia), todas almacenadas en el servidor como parte del estado del usuario.

Ejemplo

- El servidor espera que un cliente se conecte por la red. El cliente puede mandar 2 tipos de requerimientos: leer o escribir datos en un archivo. El servidor realiza la operación y retorna el resultado al cliente.
- Situación sin guardar información acerca del estado (Stateless):
  - Para leer, el cliente debe siempre especificar: nombre de archivo, posición en el archivo desde dónde debe extraer los datos y el número de bytes a leer.
  - Para escribir debe especificar el nombre completo del archivo, el lugar donde quiere escribir y los datos que quiere escribir

Ejemplo

- Situación guardardando información del estado (Stateful):
- Cuando el cliente abre un archivo se crea un entrada en la tabla. A la entrada se le asigna un *handle* para identificar el archivo y se le asigna la posición actual (inicialmente 0). El cliente recibe el *handler* como respuesta.

Handle	Filename	Posición
1	miArchivo.txt	0
2	unDocumento.doc	456
3	unJuego.exe	38
4	Hola.java	128

Ejemplo

- Cuando el cliente quiere extraer datos adicionales le envía el *handle* y la cantidad de bytes. Esto es usado por el servidor para saber gracias a la tabla de dónde exactamente debe extraer los datos (debe actualizar la posición para que para la próxima vez se pueda hacer lo mismo).
- Cuando el cliente termina la lectura/escritura envía un mensaje para sea eliminada la entrada de la tabla

Servicio REST

Servicios REST

- REST: REpresentational State Transfer (Transferencia de representación de estado)
- La clave de REST es que es stateless (sin estado)
- REST es un estilo de arquitectura, no un estándar, aunque se basa en estándares: HTTP, URL, representación de los recursos, Tipo Mime (text/xml, text/html)

Servicios REST

- No se publican servicios RPC. En arquitecturas REST, los servicios no publican un conjunto arbitrario de métodos u operaciones. Por ejemplo, en REST no podemos publicar una interfaz “IGestionEmpleados” con métodos “addEmpleado”, “removeEmpleado” o “buscarEmpleadosEnEdadDeJubilacion”.
- En REST lo que se publica son recursos. Un recurso se puede considerar como una entidad que representa un concepto de negocio que puede ser accedido públicamente. Un ejemplo de recurso sería simplemente “EmpleadosDeLaEmpresa” y otro podría ser “Empleado número 33”..

Servicios REST

- Cada recurso posee un estado interno, que no puede ser accedido directamente desde el exterior. Lo que sí es accesible desde el exterior es una o varias representaciones de dicho estado.
- La implementación del recurso decide que información es visible o no desde el exterior, y que representaciones de dicho estado se soportan. Una representación de “Empleado 33” podría ser un documento XML con la información accesible de este. Otra representación sería un documento HTML y otra podría ser un JSON.
- Podríamos pedir por ejemplo, una representación en formato imagen PNG del recurso, tal vez esto devolvería una foto del empleado, o un gráfico de su productividad o su huella dactilar.

Servicios REST

- Cada recurso tiene un identificador único global, que es su URI (o URL para los antiguos o IRI para los más modernos). Usando una URL podemos llegar a cualquier recurso en la web.
- Dada una URI, y mediante el protocolo HTTP, podemos operar sobre estos recursos. La operación a realizar se especifica mediante el verbo HTTP.
- El verbo GET hace la operación READ.
- El verbo DELETE hace la operación DELETE.
- El verbo PUT se usa normalmente para hacer UPDATE
- El verbo POST se usa normalmente para hacer CREATE.
- Las representaciones a usar se especifican mediante los llamados tipos mime. La mayoría de los tipos mime son estándares, como xml o json. El usar tipos mime estándar facilita la interoperabilidad.

Estructura URI

- {protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}
- La URI /facturas/234/editar sería incorrecta ya que tenemos el verbo editar en la misma.
- Correcta: /facturas/234
- La URI /facturas/234.pdf no sería una URI correcta, ya que estamos indicando la extensión pdf en la misma.
- Correcta: /facturas/234
- La URI /facturas/234/cliente/007 no sería una URI correcta, ya que no sigue una jerarquía lógica.
- Correcta: /clientes/007/facturas/234

SOA – Arquitectura Orientada a Servicios

Arquitectura orientada a servicios

- Forma de desarrollar sistemas distribuidos donde los componentes son servicios independientes.
- Los servicios son independientes del lenguaje y de la plataforma
- El software puede construirse al componer servicios locales y externos de diferentes proveedores
- La provisión del servicio es independiente de la aplicación que utiliza el servicio
- No es necesario decidir cuando se programa o despliega el sistema, que proveedor de servicio se debe elegir o en que servicios específicos se debe ingresar

Arquitectura orientada a servicios

- Los servicios pueden ser proporcionados localmente o subcontratan a proveedores externos
- Los proveedores pueden desarrollar servicios especializados y ofrecerlos a usuarios de servicios de diferentes organizaciones

Ejemplo: Sistema de información de un automóvil

- Un sistema de información en el automóvil proporciona a los conductores con información sobre el clima, las condiciones del tráfico, información local, etc. Esto está vinculado a la radio del coche para que la información se entregue como una señal en un canal de radio específico.
- El vehículo está equipado con un receptor de GPS para descubrir su posición y, sobre la base de esa posición, el sistema accede a una amplia gama de servicios de información. La información puede ser entregada en el idioma especificado del conductor.

Estándares SOA

- SOAP. Simple Object Access Protocol es un protocolo de mensajería para el intercambio de información entre sistemas. A diferencia de otros estándares de comunicación por mensajes (CORBA, RMI) se basa en documentos XML, por lo que resulta independiente de la tecnología y la plataforma (lenguaje, sistema operativo) subyacente
- WSDL. Web Services Definition Language es un lenguaje XML para describir los servicios en la arquitectura SOA.
- UDDI. Universal Description, Discovery and Integration define los mecanismos para publicar (catalogar) servicios en un registro, y para que sus consumidores puedan buscarlos y encontrar su localización física

Desambiguando términos

SOA vs Servicios Web

- Con SOA, toda la infraestructura de tecnologías de la información (TI) presenta sus funcionalidades como servicios que ofrecen un claro valor de negocio
- SOA involucra servicios web
- Pero un servicio web puede ser utilizado en otro estilo arquitectónico

Servicios vs Componentes

- No toda entidad que posea interfaz, lógica y ofrezca una funcionalidad es un servicio
- Servicios manejan conceptos de negocio (↑ nivel abstracción, ↓ granularidad)
- Servicios accesibles por protocolos abiertos e independientes de la implementación
- Servicios están menos acoplados que los componentes
- Servicios con mayor grado de autonomía (tienen sentido incluso de forma aislada)
- Servicios tienen sus propias políticas de escalabilidad, seguridad, tolerancia a fallos, etc.

ARQUITECTURA DE SOFTWARE ORIENTADA A SERVICIOS

Autor: Luciano Straccia