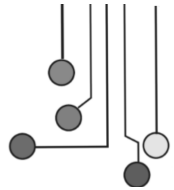


Rendimiento y seguridad

Sistemas de Información Orientados a Servicios

RODRIGO SANTAMARÍA

- Invocación
- Monitorización
- Ejemplos
- Reusabilidad y compatibilidad
- Requisitos
- Interoperabilidad
- Múltiples clientes
- Heterogeneidad
- Múltiples capas
- Rendimiento y Estado
- XML y servicios web
- Ejemplos de ataques

Rendimiento y seguridad

Rendimiento

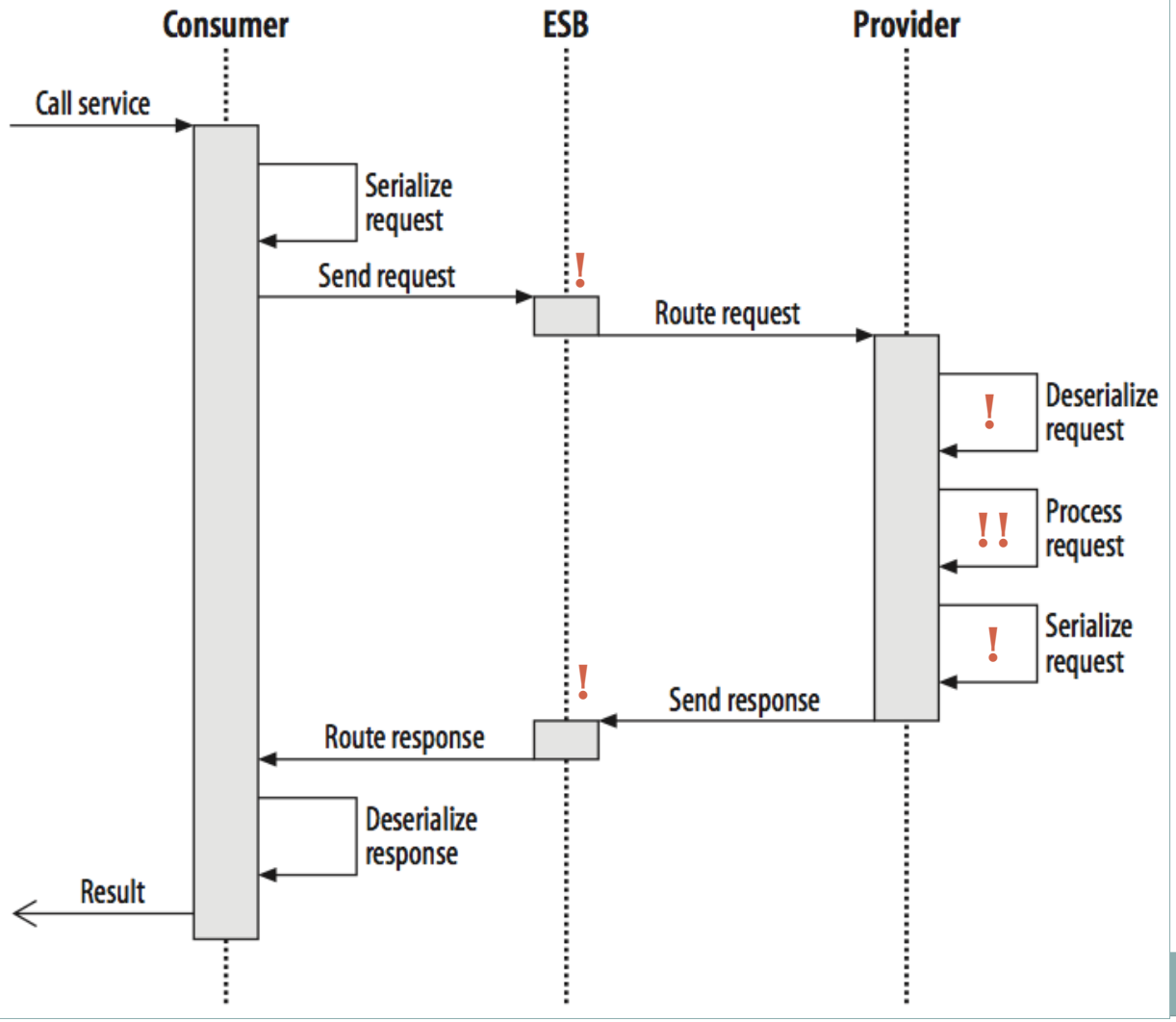
INVOCACIÓN
MONITORIZACIÓN
EJEMPLOS

RENDIMIENTO Y REUSABILIDAD
RENDIMIENTO Y COMPATIBILIDAD

Rendimiento

- Puntos críticos para el rendimiento
  - Tiempo de ejecución de un servicio
    - La causa más común
    - Bien por implementación o por uso de recursos/cuellos de botella
    - Importante: incluir en los SLA
      - Tiempo medio de respuesta por invocación
      - # de invocaciones que pueden servirse en un periodo de tiempo
  - Serialización y ancho de banda
    - Sobre todo si ESB debe hacer mapeos entre protocolos o para verificar la corrección de un mensaje

Rendimiento: invocación



- Invocación normal síncrona
- Puntos críticos
  - Envío de petición/respuesta por ESB
  - Serialización de la petición o la respuesta
    - XML (x4 a x20 en tamaño), aunque normalmente no es un problema
  - Procesamiento de la petición
    - El problema más común

Rendimiento: monitorización

- Es difícil predecir qué puntos resultarán críticos
  - Depende del uso, las redes, los parámetros de entrada, etc.
  - Necesidad de monitorización de los tiempos de ejecución
- A continuación veremos algunos ejemplos de cómo un procedimiento aparentemente eficiente puede dar problemas de rendimiento
  - Paso de un procedimiento remoto a un servicio
  - Rendimiento y reusabilidad
  - Limitaciones en la invocación
  - Servicios a medida

Ej. 1: de procedimientos remotos a servicios

- Sea un procedimiento para acceso remoto a una BD
  - Eficiente (entre 0.01 y 0.03 s)
  - La empresa lo ha estado usando según escalaba y ha perdido el control de qué sistemas lo están usando
  - Modificar el esquema de la BD es arriesgado y costoso, pues podría hacer fallar muchos sistemas
- Solución: en vez de usar la BD a través del acceso remoto, ahora los consumidores deberán hacerlo mediante un servicio (para poder cambiar la BD)
  - El servicio funciona con tiempos entre 0.1 y 0.3 s, tiempo suficiente para parecer inmediato en interfaces gráficas (<0.5s)

Ej. 1: de procedimientos remotos a servicios

- Aunque en las pruebas unitarias no se detectaron problemas, el servicio es entre 5 y 10 veces más lento que el procedimiento remoto
  - Los consumidores que hacen 10+ invocaciones antes tenían tiempos de respuesta entre 0.1 y 0.3 s, ahora son entre 1 y 3 s.
- **Solución**: añadir llamadas en paralelo
  - Mucho más complejo que lo planeado: clientes más complejos para manejar llamadas asíncronas, concurrencia en proveedor, etc.

Ej. 2: rendimiento y reusabilidad

- Qué significa granularidad ‘gruesa’ depende muchas veces del rendimiento, y es un equilibrio:
  - Evitar llamadas de grano fino que en suma sean menos eficientes que una sola de grano grueso → juntar
  - Evitar el grano grueso de una sola llamada que procese muchos datos si se vuelve costosa → dividir

Ej. 2: rendimiento y reusabilidad (II)

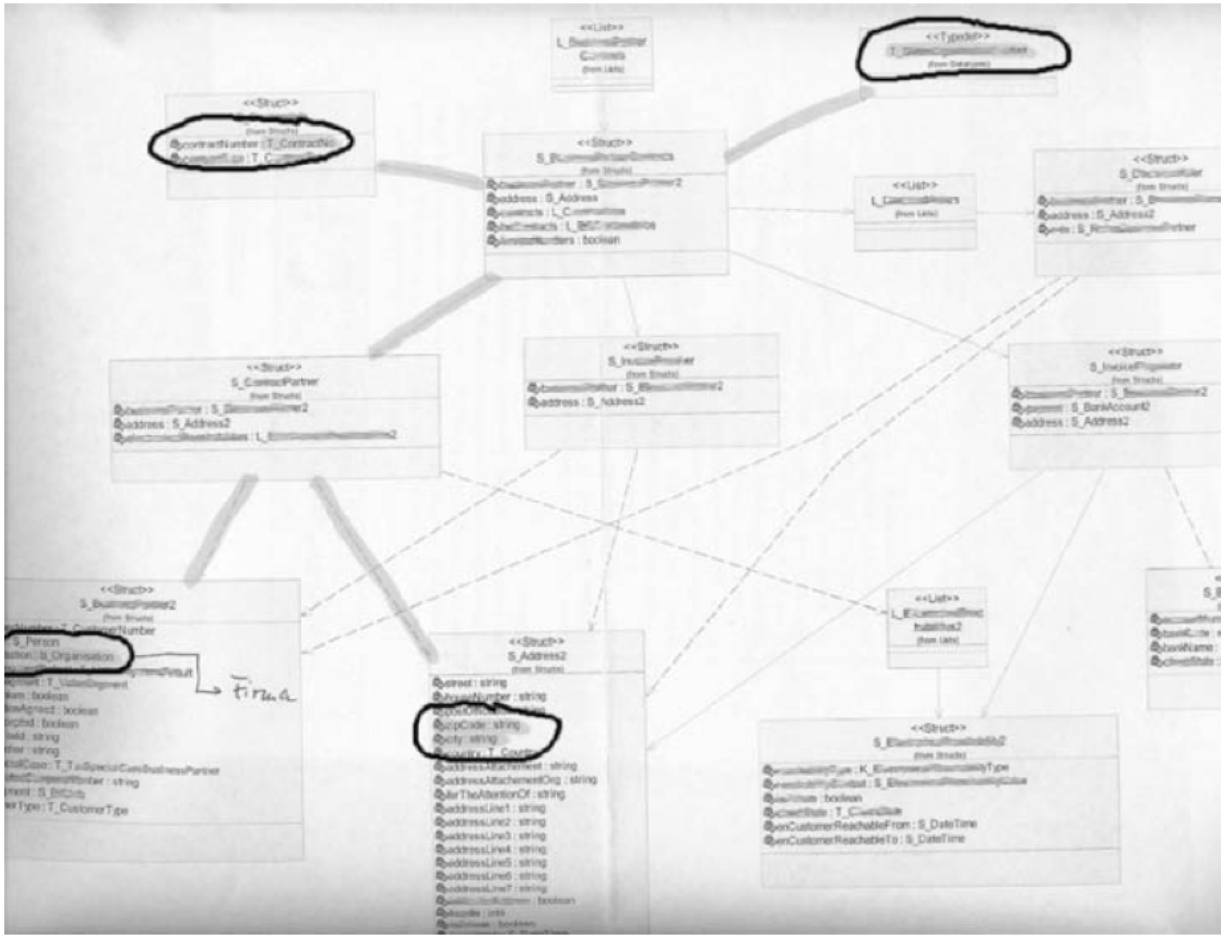
- Sea un servicio *S1* ofrecido por el sistema CRM* de una compañía que retorna todos los datos del cliente
  - 'todos los datos' puede incluir cientos de atributos:
    - ID, direcciones asociadas, métodos de pago registrados
    - Todos los contratos, recibos y pagos registrados
- Un nuevo cliente pide un servicio que retorne los datos de sus clientes
  - En aras a la reusabilidad, el equipo CRM recomienda usar *S1*
  - El consumidor se queja porque *S1* es muy lento ya que sólo necesitan 5 atributos → solución: crear un nuevo método *S2*

*CRM – customer relationship management, gestión de relaciones con el cliente

Ej. 2: rendimiento y reusabilidad (III)

- Los 5 atributos se encuentran dispersos!
  - El método reusable no era eficiente
  - El nuevo método más simple va a requerir esfuerzo
- Esto no quiere decir que la reusabilidad sea imposible o inadecuada,
  - pero sí que no siempre va a ser eficiente reutilizar

[Josuttis07, fig 13.4]



Ej. 3: limitaciones en la invocación

- Para estimar el rendimiento hay que monitorizar
  - Sólo se puede monitorizar si el servicio está en producción
- Sea un servicio *S* que se proporciona a dos clientes
  - En las etapas finales del desarrollo, vemos que hay que dividir el servicio en dos por problemas de rendimiento
    - Añadir un nuevo servicio en fases tardías puede ser un problema
- Solución: limitaciones en la invocación
  - Añadir a *S* un atributo adicional `callConstrains`
    - Si un cliente sufre problemas de rendimiento, este atributo actúa como un flag para hacer algunas optimizaciones/casos especiales
    - Es en el fondo un 'rodeo', pero puede ser útil

Ej. 3: limitaciones en la invocación

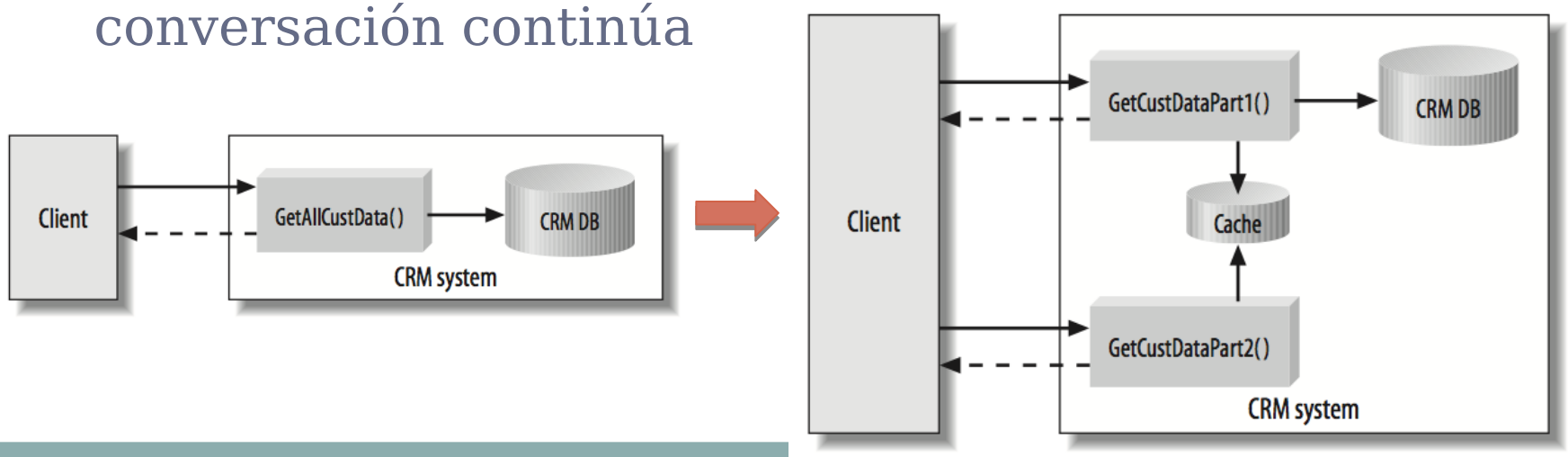
- Podemos añadir atributos con notación más flexible
  - Pero demasiada flexibilidad puede ser perjudicial
    - Parseo y procesamiento del atributo(s)
    - Verificación de semántica correcta
    - Mantenimiento más complicado
    - Servicio con un formato más complejo para el cliente
- Punto medio:
  - Evitar que el servicio se vuelva genérico
  - Evitar que el servicio devuelva distintas estructuras
  - Un servicio debe hacer una funcionalidad de negocio concreta

Ej. 4: servicios a medida

- Sistemas diferentes → concepción diferente de los mismos datos → servicios diferentes para mantener el rendimiento
- A veces, incluso es necesario crear servicios no sólo distintos sino **específicos** para un consumidor

Ej. 4: servicios a medida

- Ejemplo: sea un sistema CRM y un servicio (lento) que carga todos los datos de un cliente
  - El operador quiere todos los datos, pero quiere algunos de ellos (nombre, contraseña, etc.) inmediatamente para poder iniciar la conversación, y que el resto se vayan cargando mientras la conversación continúa

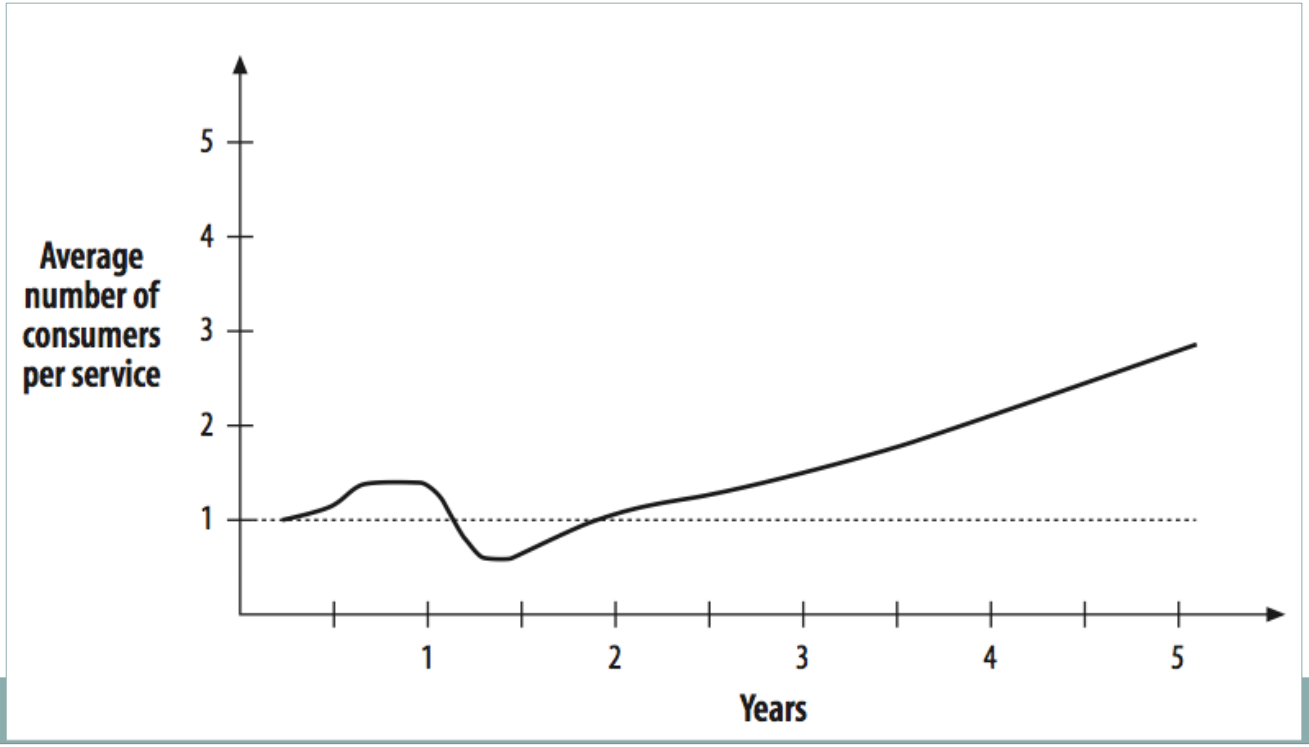


Rendimiento y reusabilidad

- Los ejemplos anteriores muestran que el rendimiento puede reducir la reusabilidad
  - En teoría: un método para retornar todos los datos del cliente
  - En la práctica: varios métodos para distintas vistas (incluso a medida)

Rendimiento y reusabilidad

- Podemos estar satisfechos si en un contexto SOA cada servicio, al cabo de 1-2 años, lo usa un consumidor
  - Y al cabo de 3-4 años, 2+ consumidores
  - Hay servicios que puede no usar nadie!
    - Se implementaron pero el consumidor no los encontró útiles finalmente
    - El consumidor puede haber cambiado a otros servicios



[Josuttis07, fig. 13.7]

Rendimiento y compatibilidad

- Volvamos al ejemplo del servicio que retorna datos de clientes en una compañía telefónica
  - Generalmente, estas compañías hacen una distinción entre clientes de empresa y particulares
  - Sin embargo, nuestra compañía tiene particulares que consumen mucho y empresas que consumen poco, y además no queda claro cómo tratar a los autónomos
    - Solución: añadir un atributo ‘volumen de llamadas’ que va de 1 a 10 y extender los servicios con un atributo numérico
    - Añadir un atributo sencillo no debería afectar al comportamiento de los servicios existentes: es compatible hacia atrás

Rendimiento y compatibilidad (I)

- En la práctica, estas nuevas versiones de los servicios elevaban el tiempo de ejecución en un factor >=2
  - La nueva versión no era útil así para el cliente
- ¿Por qué?
  - El cálculo del atributo ‘volumen de llamadas’ se estaba haciendo en tiempo de ejecución ya que podía haber clientes con más de un número contratado → peor rendimiento
    - El servicio firmado funcionaba, pero el contrato acordado (90% de las llamadas respondidas en menos de 0.5 s) se había roto
- Es claramente un error de diseño, se soluciona calculando el atributo en batch… ¿o no?

Rendimiento y compatibilidad (II)

- Este ejemplo ilustra algunas lecciones:
  - En sistemas grandes y complejos, es **imposible anticipar todos los posibles errores**, oscureciendo aspectos obvios
  - **No existe solución perfecta**: en este caso, calcular en batch los millones de clientes requeriría muchos más recursos de memoria que calcular en tiempo de ejecución las decenas de miles consultados diariamente.
  - Incluso si se reconoce el error y se implementa una solución, puede llevar a **involucrar a nuevos sistemas** (p. ej. el departamento responsable del mantenimiento de la BD que tiene que almacenar los atributos procesados en batch)

Seguridad

REQUISITOS
INTEROPERABILIDAD
MÚLTIPLES CLIENTES
HETEROGENEIDAD
MÚLTIPLES CAPAS
RENDIMIENTO Y ESTADO

XML Y SERVICIOS WEB
EJEMPLOS DE ATAQUES

Requisitos de seguridad

- Autenticación
  - Identificar quién está llamando al servicio
- Autorización
  - Determinar si el invocador tiene permitido invocar al servicio o ver el resultado
- Confidencialidad
  - Asegurar que nadie aparte del invocador puede ver datos del servicio transferidos entre proveedor y consumidor
- Integridad
  - Garantizar que los datos no han sido manipulados

Requisitos de seguridad

- Disponibilidad
  - Garantizar que el servicio está disponible (ataques DoS, etc.)
- Registro
  - Seguimiento de las invocaciones del servicio para gestión, planificación, facturación, etc.
- Auditoría
  - Registro de toda la información relevante en cuanto a seguridad para detectar o analizar agujeros de seguridad o ataques
  - AAA: autenticación + autorización + auditoría

Seguridad y SOA

- Mismas aproximaciones que en sistemas distribuidos
  - Autenticación y autorización
    - IDs de usuarios y constraseñas + roles/privilegios
    - Otros estándares (Kerberos, certificados, etc.)
  - Confidencialidad e integridad
    - Encriptación, firma digital
- Sin embargo, hay algunos aspectos que pueden influir en la seguridad
  - Interoperabilidad
  - Heterogeneidad

Seguridad vs Interoperabilidad

- Si queremos interoperabilidad, nuestro sistema tiene que estar abierto
  - “Tenemos dos murallas, un foso, flechas y aceite hirviendo, pero la interoperabilidad alta requiere tener el puente levadizo bajado” –Bruce Sams (OPTIMA, www.optimabit.com)
- Para proteger datos sensibles
  - Restringir la capacidad del consumidor para invocar servicios *y* ver resultados

Seguridad vs Interoperabilidad (II)

- Por ejemplo, casi todo servicio de información requiere hoy en día un paywall
  - Pero también requiere que un buscador lo indexe.
  - Los crawlers de los buscadores requieren leer sitios sin javascript para ser eficientes
  - En el javascript suele estar el paywall, así como toda la 'basura' de anuncios a la que estamos desgraciadamente acostumbrados
- Problema (o solución, según se mire)
  - Cliente que se hace pasar por crawler: https://12ft.io

Múltiples clientes

- Separar la capacidad de invocar a un servicio de la capacidad de ver ciertos resultados
  - Ejemplo: un sistema de procesamiento maneja datos de varios bancos
    - Todos los bancos acceden al mismo servicio de consulta de clientes
    - Pero no deben poder obtener datos de clientes de otros bancos
    - Parece algo sencillo → limitar acceso por el número de cuenta

Múltiples clientes

- Permitir sólo cierto tipo de invocaciones estáticamente puede no ser suficiente
  - Ejemplo: como antes, pero con compañías telefónicas
    - Proveedores de servicio y de red distintos
    - Algunos datos pueden mantenerse privados, pero otros deben poder ser accedidos por todos los proveedores
      - Por ejemplo, para permitir la portabilidad de números
    - No es suficiente con limitar el acceso en función del número/red
- Necesidad de efectuar chequeos de seguridad en tiempo de ejecución

Seguridad vs Heterogeneidad

- Introducir un concepto general de seguridad sobre muchos conceptos de seguridad distintos
  - Uno en cada sistema: distintos IDs de usuario para la misma gente, etc.

Múltiples capas

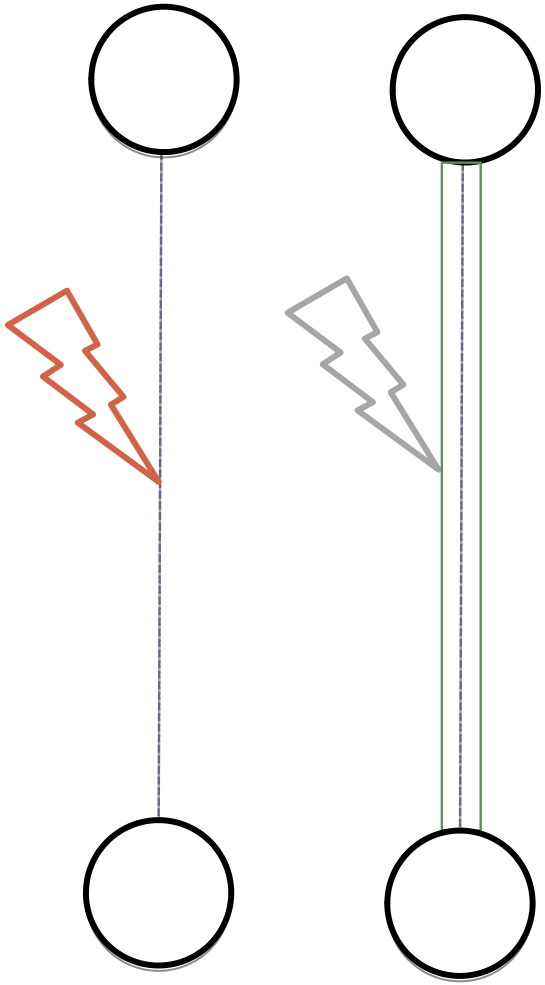
- Seguridad punto a punto
  - Desde el consumidor (frontend) hasta el proveedor (backend)
  - Seguridad a nivel de transporte
  - P. ej. Secure Sockets Layer (SSL) disponible en Internet y Servicios Web (HTTPS)
  - No es suficiente! Ejemplo*
    1. Imaginemos que Facebook tiene una API con un servicio “publicar en el muro” que requiere que el usuario esté autenticado
    2. Estando autenticados en Facebook, accedemos a badsite.com
    3. badsite.com aprovecha la API de Facebook y envía un mensaje
    4. El navegador la envía con nuestra información de usuario y funciona: postea en nuestro nombre un anuncio de badsite.com

* Respuesta 3 (Jason P) en http://stackoverflow.com/questions/21885765/cross-origin-resource-sharing-cors-and-javascript

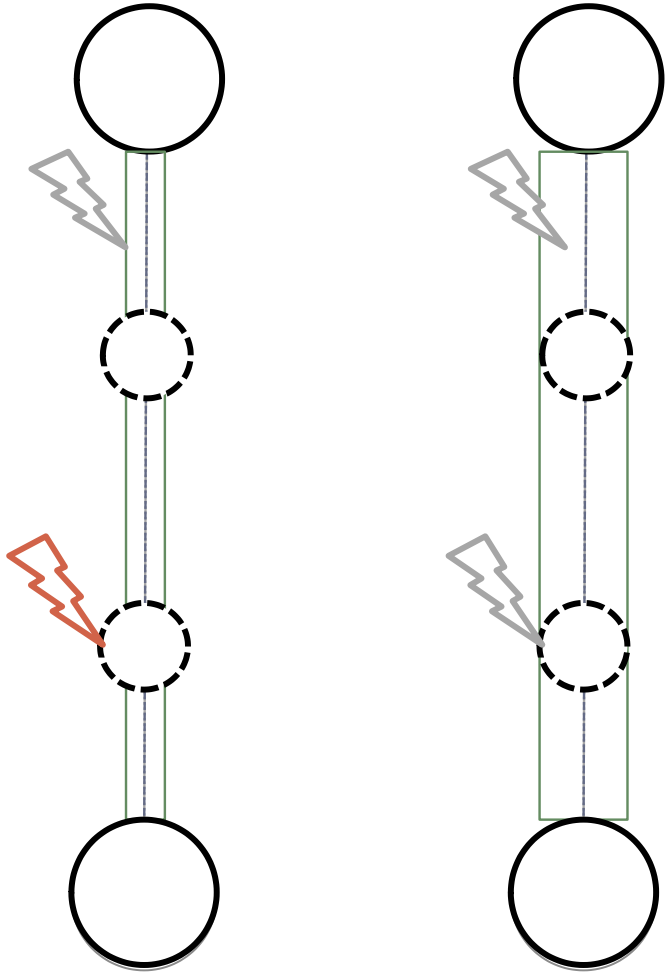
Múltiples capas

- Seguridad punto final a punto final
  - Seguridad a nivel de mensaje: el propio mensaje incorpora directivas de seguridad
  - P. ej. Cross Origin Resource Sharing (CORS)*
    - Para evitar problemas como el anterior, los navegadores son muy estrictos, bloqueando la invocación al servicio o el acceso a la respuesta (same-origin policy)
    - CORS permite añadir una cabecera de 'control de acceso' al mensaje con la URL de modo que el navegador sólo entrega el mensaje a la aplicación con la misma dirección desde la que se invocó

* http://mortoray.com/2014/04/09/allowing-unlimited-access-with-cors



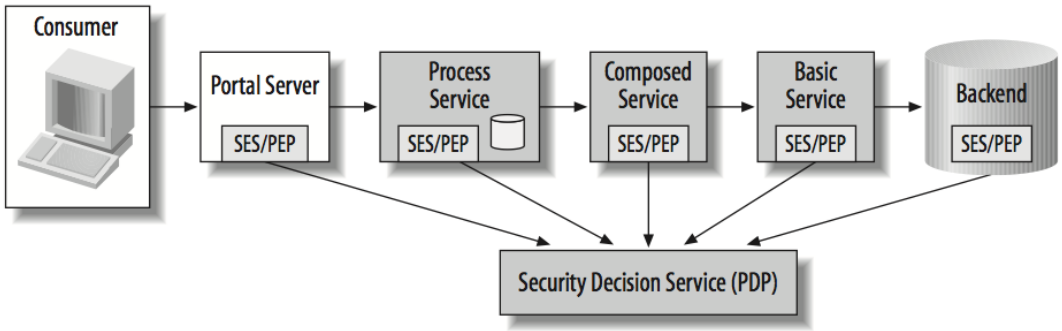
Seguridad punto a punto



Seguridad punto final a punto final

Seguridad como servicio

- Puntos de Decisión de la Política (PDP)
  - Se encargan del control de acceso
  - Generalmente en servicios, servidores o backend
- Puntos de Refuerzo de la Política (PEP)
  - Refuerzan o revisan el control de acceso garantizado
  - Opciones
    - En consumidor (gracias a la seguridad a nivel de mensaje)
    - En servicios especiales para refuerzo de seguridad que se invocan desde la aplicación o el ESB



Seguridad, rendimiento y estado

- La seguridad tiene asociado un coste en rendimiento
  - Encriptación de mensajes, validación de la integridad, etc.
  - La alternativa es poner el sistema en peligro
  - → Equilibrio entre riesgo y rendimiento
- La seguridad afecta al estado
  - Los servicios *sin estado* se vuelven *con estado* al tener en cuenta información de acceso del usuario (que normalmente no se transfiere con cada invocación del servicio)
    - Cachés renovadas cada cierto tiempo (¿cuánto?)

Seguridad en la práctica

- La seguridad se implementa de manera heterogénea
  - Autenticación, encriptación por capas, etc.
- A menudo hay áreas en las que no hay seguridad
  - Es muy difícil cubrir todas las posibles violaciones
- Lo más importante es trazar y correlacionar flujos de datos
  - Registro, monitorización, auditoría del tráfico de mensajes
  - Desincentiva al posible atacante
  - Al menos permite identificar las violaciones que ocurren
  - Uno debe determinar si optar sólo por esta solución es un riesgo muy alto o no

"Al menos 50' millones de cuentas de Facebook hackeadas" (sep 2018)
https://www.eldiario.es/tecnologia/millones-cuentas-Facebook-hackeadas_0_819269005.html

Seguridad en la práctica: XML y WS

- Existen estándares de seguridad en Servicios Web y XML
  - Similares a otras tecnologías o procedimientos
- Para XML
  - Firma: permite firmar documentos para asegurar integridad
  - Encriptación: total o parcial
  - Manejo de claves

Seguridad en la práctica: WS y XML

- Para Servicios Web
  - **WS-Security**: describe cómo introducir la información de seguridad (firmas, Kerberos, encriptación, etc.) en la cabecera del mensaje
  - **WS-Policy**: permite explorar/exigir la política de seguridad asociada a un servicio
  - **WS-Trust**: para petición, renovación o validación de certificados de seguridad
  - **WS-SecureConversation**: establece un contexto compartido de seguridad para varios intercambios de mensajes, en aras a la mejora de rendimiento

Ataques en SW y XML: Bombas XML

- Los parseadores de XML pueden expandir documentos XML pequeños en textos grandes
  - El ejemplo de la derecha expande a 3x3x3 elementos
  - Con 10 expansiones de 10 elementos, tendríamos 10^{10} elementos, lo que seguramente bloquee la memoria y haga fallar al proceso
- Una especie de DoS no detectable por un firewall, pues el problema no es el nº de mensajes sino el contenido

```
<?xml version="1.0"?>
<!DOCTYPE xmlbomb [
<!ELEMENT data (#PCDATA)>
<!ENTITY a "&b;&b;&b; ">
<!ENTITY b "&c;&c;&c; ">
<!ENTITY c "&d;&d;&d; ">
<!ENTITY d "foo ">
]>
<data>&a;</data>
```

<data>foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo</data>

Ver Billion Laughs Attack, que afectó a Kubernetes todavía en 2019

Ataques en SW y XML: Inyecciones XPath

- XPath es un lenguaje para construir expresiones de consulta y procesamiento de un XML
  - Similar a SQL para bases de datos
- Las inyecciones XPath, como las SQL, implican introducir código malicioso (generalmente en campos de texto libre) para hacer órdenes distintas
  - `SELECT * FROM usuarios WHERE nombre = 'Alicia';`
  - Sustituir `'Alicia'` por `'Alicia'; DROP TABLE usuarios; SELECT * FROM datos WHERE nombre LIKE '%'`
    - El código en rojo se ha 'inyectado' y borrará la tabla `usuarios` y accederá a la tabla `datos` que no debería ser accesible

La inyección de código había caído por debajo del 1% de todas las vulnerabilidades detectadas para 2016, pero está volviendo a repuntar: https://nvd.nist.gov/vuln/search

Ataques en SW y XML: SOAP y REST

- XML permite Document Type Definition (DTD)
  - Dentro de un XML podemos tener información, por ejemplo, de rutas de archivo
  - En SOAP, los DTDs no están permitidos, pero en REST no hay restricciones a priori
- Adjuntos SOAP
  - Un mensaje SOAP puede contener adjuntos, que pueden ser grandes y/o difíciles de procesar (DoS) o contener virus
    - Deben ser a) bloqueados, b) filtrados a no ser que sean tipo MIME o c) escaneados por un antivirus

Ataques en SW y XML: Captura-reenvío

- Un servicio en SOA que sólo acepta mensajes firmados puede dar acceso no autorizado a un atacante que capture un mensaje firmado y lo reenvíe
  - WS-Security permite incluir marcas temporales
  - WS-Policy permite exigir un timestamp firmado
  - Se acepta un mensaje con timestamp tras un tiempo que sea
    - Lo suficientemente pequeño para que al atacante no le dé tiempo a capturar, desencriptar y reenviar
    - No tan pequeño como para causar problemas por las diferencias entre los relojes de los sistemas

Seguridad en SW: API key y Access Token

- Una **API key** es un código único por el que identificamos la *aplicación* que invoca el servicio
  - Permite identificar al posible asaltante.
- Un **Token de Acceso** es un código por el que identificamos al *usuario* que invoca el servicio, a través de una sesión o invocación única.
  - Evita problemas de duplicación o DoS
  - Identifica usuario, permisos, etc.

Seguridad en SW: JSON Web Token (JWT)

- Estándar (2015) para identificar/encriptar una sesión en JSON
- Consta de tres partes:
  - **Cabecera**: indica el algoritmo de encriptación (`alg`) y el tipo de token (`typ`)
  - **Carga** (payload): indica información de seguridad, p. ej.:
    - cuándo se lanzó la petición (*issued at* - `iat`)
    - nombre de usuario
  - **Firma**: hash(cabecera).hash(carga)
- Las tres partes se codifican con base 64 y se concatenan
  - base64(cabecera).base64(carga).base64(firma)

Base 64

- Esquema de conversión binario → texto
- Usado en WWW para transmitir imágenes o archivos binarios por HTTP (texto)
  - También para transmitir texto evitando problemas de codificación o internacionalización.
- Usa un alfabeto que define letras (A-Z a-z), dígitos (0-9) y dos caracteres (+ y /) mediante 26=64 bits (26+26+10+2).
- Ejemplo: *Man* (ASCII) → *TWFu* (Base64)

Source	Text (ASCII)	M							a								n							
	Octets	77 (0x4d)							97 (0x61)								110 (0x6e)							
Bits		0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Base64 encoded	Sextets	19							22						5			46							
	Character	T							W						F			u							

Fuente: Wikipedia

Resumen

- El **rendimiento** es un problema en el caso de servicios, especialmente en cuanto a **tiempos de ejecución**
- Cambiar de procedimientos remotos propios a servicios puede ser un problema
- Puede haber un compromiso entre **reusabilidad y rendimiento**
- El rendimiento puede tener un impacto en la **granularidad**, los servicios tienden a hacerse de grano grueso si hay mucha carga por múltiples invocaciones de servicios de grano fino, pero no deberían hacerse tan gruesos que se vuelvan genéricos
- Se pueden utilizar **limitaciones en las llamadas y atributos no funcionales** (SLAs) para mejorar el rendimiento, aunque puede dar problemas de compatibilidad
- SOA tiene los problemas de **seguridad** similares a cualquier sistema distribuido o aplicación web, con la peculiaridad de que debe tratar con **interoperabilidad alta**, que reduce significativamente la seguridad
- Las soluciones punto a punto no son suficientes y **necesitamos soluciones punto final a punto final**, que normalmente ntroducen conceptos de seguridad en los propios mensajes
- La seguridad afecta al **rendimiento** y al **estado** de los servicios. Es bueno tenerla en cuenta, pero es difícil cubrir todos los aspectos y a veces la solución de compromiso es mantener una buena **monitorización**
- **XML** y los **Servicios Web** introducen problemas similares a los que podemos tener en aplicaciones web, aunque erróneamente se tratan de manera más ligera

Referencias

- [Josuttis07] Nicolai M. Josuttis. *SOA in practice. The Art of Distributed System Design*. O'Reilly, **2007**.
  - Ch 13 (rendimiento)
  - Ch 14 (seguridad)