Sistemas distribuidos e integración

Preparado por: Luis Daniel Benavides Navarro, Ph.D.

Modularidad por medio de clientes y servicios

Problema

- Aprendimos que dividir un sistema en módulos es una buena estrategia de diseño

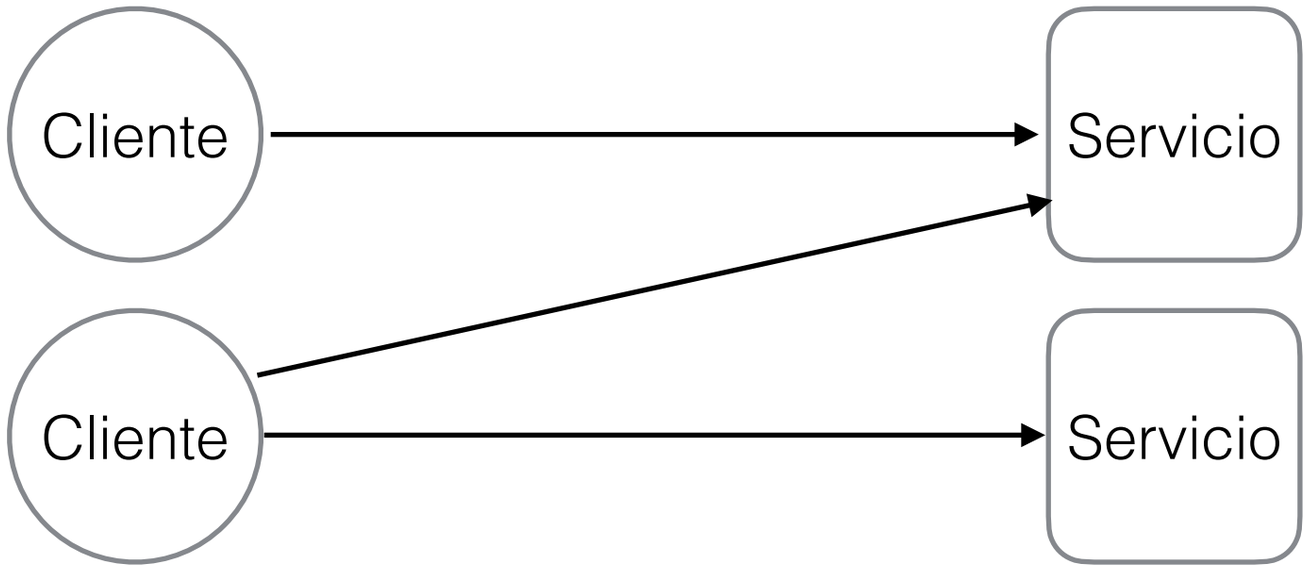
- Si los programadores/Diseñadores no cometieran errores esto es todo lo que necesitaríamos.

- La modularización no impide que se propaguen errores en el sistema. Hay interacciones implícitas inesperadas.

- Ante el error necesitamos formas más fuertes de modularización que protejan el sistema de errores.

- Organizar el sistema con clientes y servicios protege el sistema de interacciones inesperadas.

Clientes y Servicios



Al menos tres beneficios:

- Los mensajes son la única forma de interacción (limita interacción y protege contra violación de modularidad)
- Los mensajes son la única forma en que los errores se propagan
- Los mensajes son la única forma en que un atacante penetra un servicio

Esta simple estratégia permite crear sistemas modulares, tolerantes a fallas y seguros.
Es LA estrategia primordial para diseñar sistemas complejos.

Algunas consideraciones de diseño

- La división en funciones es una forma débil de modularidad. Ya que las funciones comparten memoria y pueden crear interacciones inesperadas y propagar errores.

- Por ahora consideremos que cada cliente y cada servicio corre en su propio computador y se comunican por un cable.

- Ejemplo: Aplicación Web con al menos dos servicios. Al menos dos opciones: Un servidor con dos métodos, o dos servicios independientes.

Mecanismos de integración

Puntos de decisión para la integración de aplicaciones

- Acoplamiento de la aplicación
- Intrusividad: minimizar cambios en aplicación y en mecanismo de integración
- Selección de tecnología
- Formato de los datos

Puntos de decisión para la integración de aplicaciones (cont.)

- Tiempo de vida de los datos
- Datos o funcionalidad
- Comunicación remota: Sincronía, parámetros, fallos etc.
- Confiabilidad

Estrategias de integración

- Transferencia de Archivos
- Base de datos compartida
- Invocación remota de métodos
- Mensajería

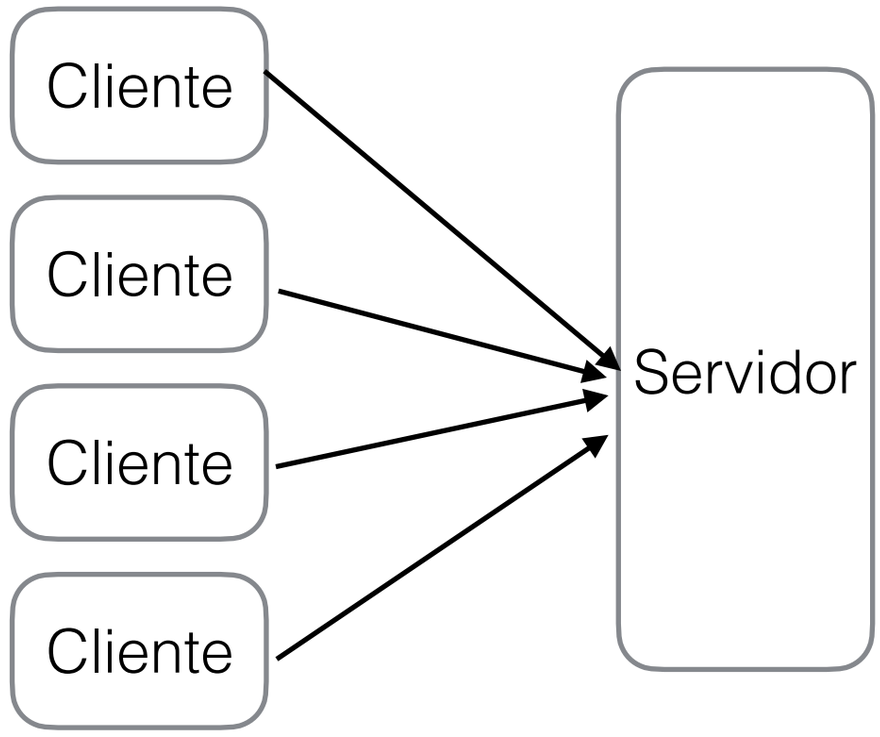
Patrones de integración

Patrón 1: Remote procedure call (RPC)

- Solicito un servicio y espero por la respuesta
- Simula un llamado local de procedimiento
- Pero no lo es
    - La semántica de invocación es diferente (debe prepararse para los errores y considerar el marshaling)
    - Se demora más tiempo la respuesta
- Ejemplos
    - Bases de datos

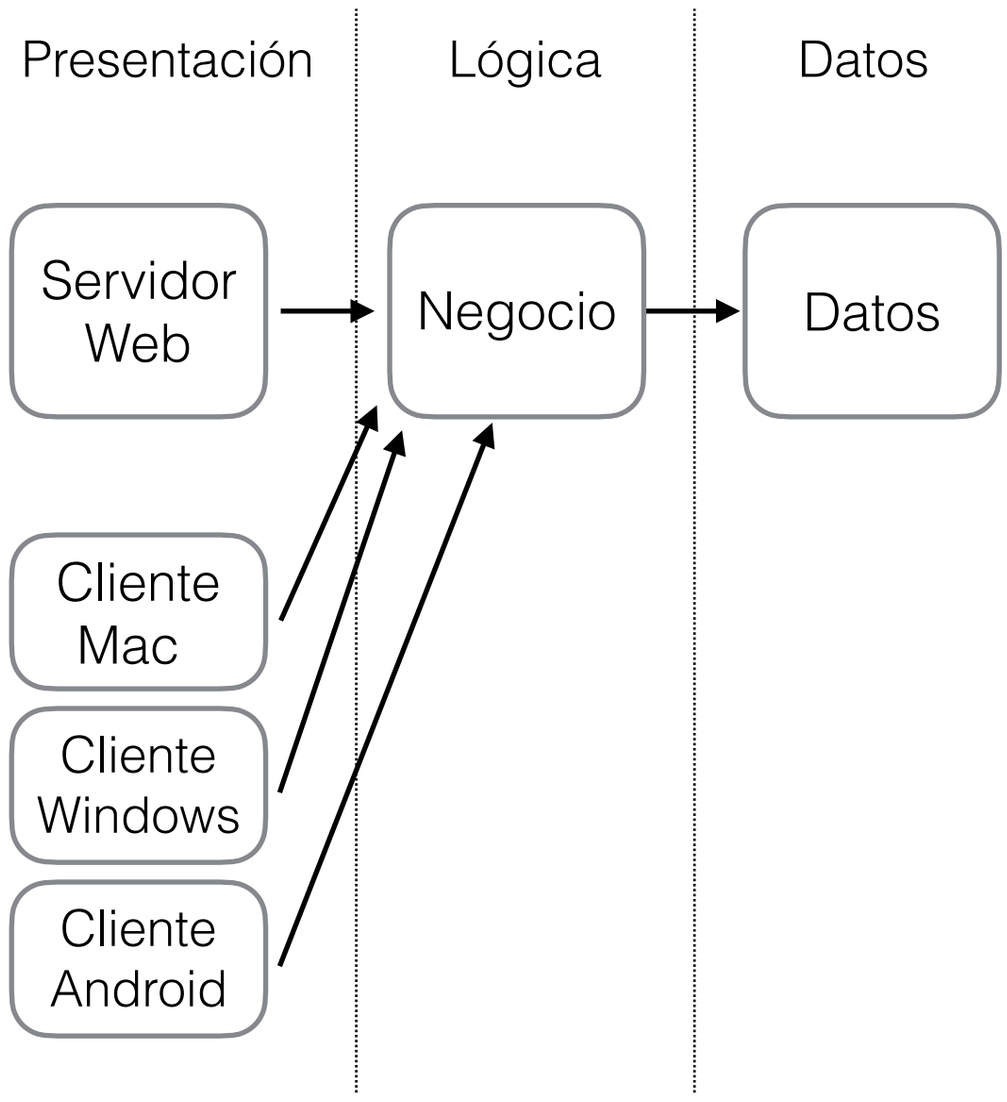
Patrón 2: Cliente-Servidor

- Múltiples clientes generalmente iguales, un servidor
- Arquitectura simple
- Hay que actualizar todos los clientes cuando hay un cambio
- Servidor único punto de falla
- Ejemplo: App conectada a base de datos



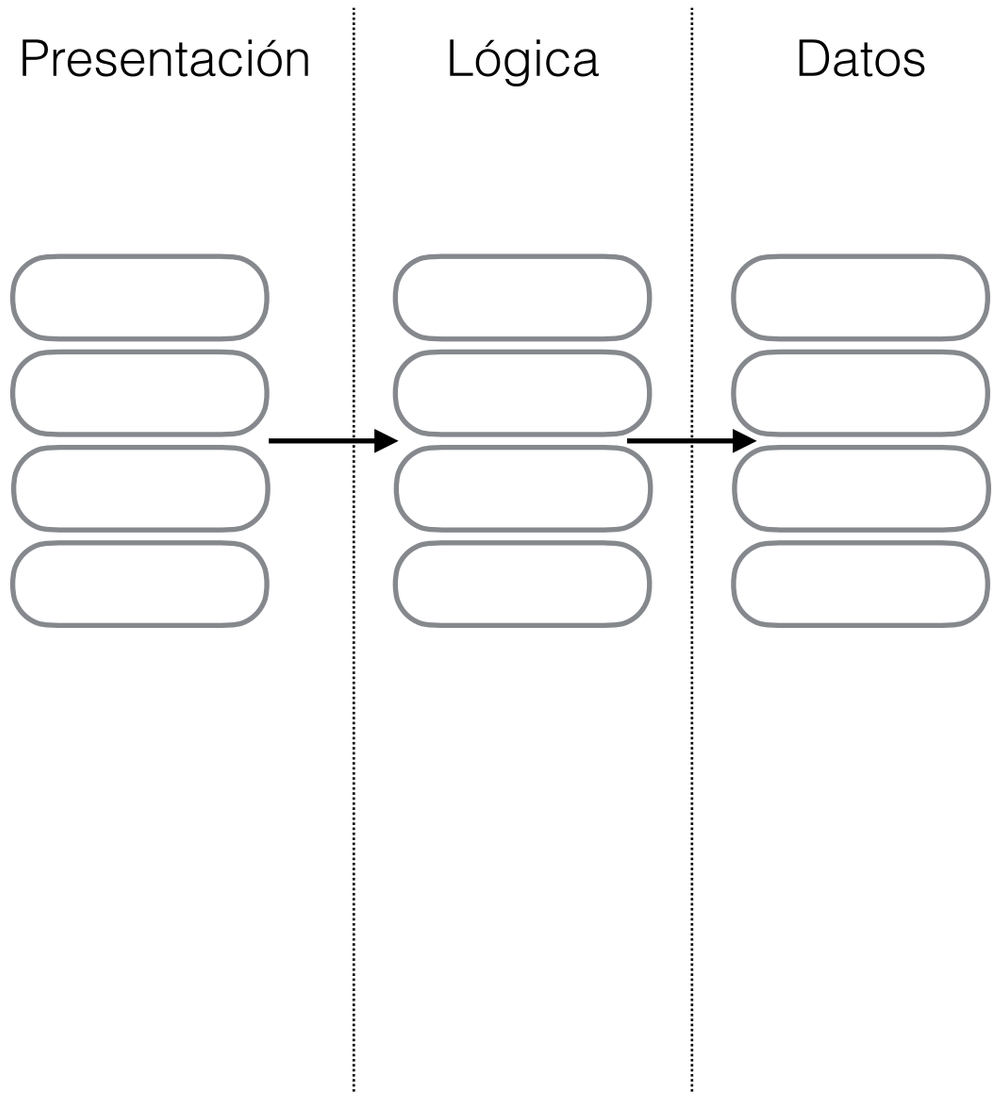
Patrón 3: 3 Capas

- Se separa en tres capas típicas: Presentación, lógica, persistencia
- Arquitectura simple
- Facilidad para soportar múltiples clientes
- Cuándo se actualiza la lógica no es necesario actualizar clientes.
- Ejemplo: Evernote



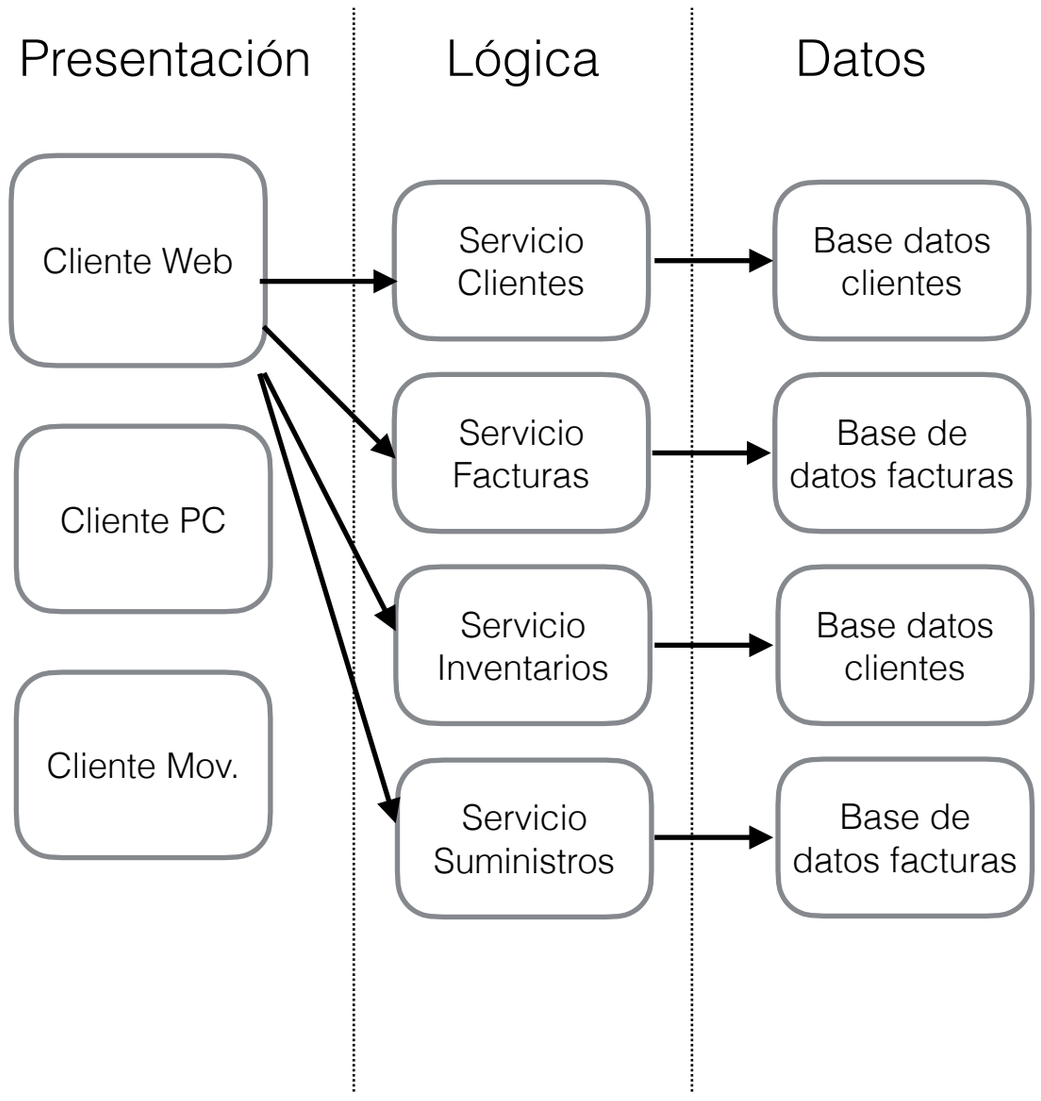
Patrón 4: Clustering

- Separación en capas
- Alta disponibilidad
- Mejor desempeño
- Permite estrategias elásticas
- Ejemplo: Evernote en AWS



Patrón 5: Microservicios

- Cada servicios es autónomo
- No es monolítico
- Facilidad de desarrollo y despliegue
- Alta complejidad
- Coordinación compleja
- Permite ultra-escalabilidad
- OJO con la granularidad del servicio!!
- Ejemplo: Netflix, Amazon, Facebook

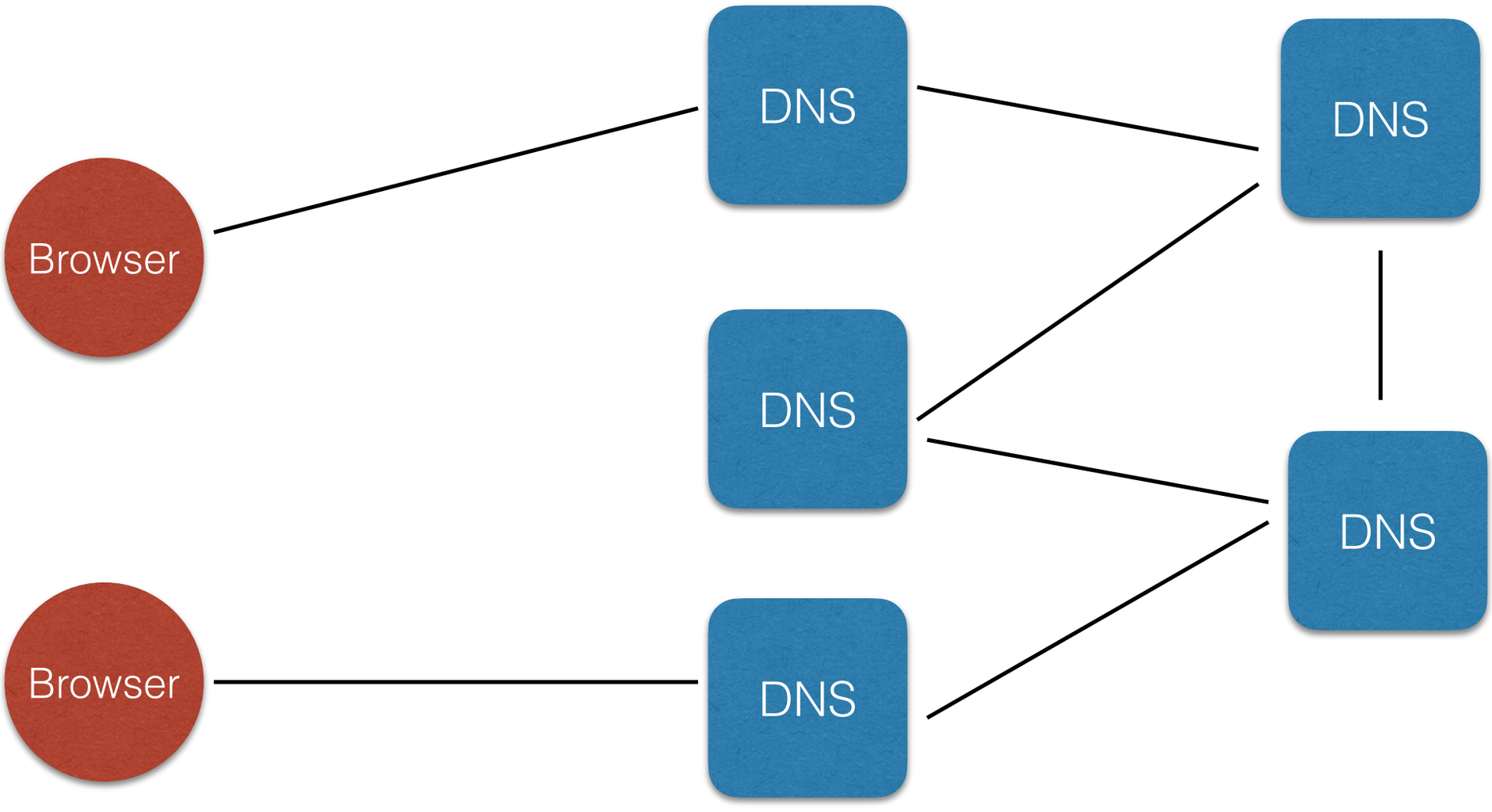


Patrón 6: Intermediario

- Comunicación por medio de un intermediario
- Permite patrones de comunicación asíncronos
  - Push vs. Pull
  - Desacoplar con indirección (Por ejemplo, enviar a decano@escuelaing.edu.co)
  - La indirección permite decidir cuando duplicar mensajes (Ej, enviar a lista de correos)
  - **Publish subscribe**
- Ejemplos:
  - Email, SOA, MENSAJERIA

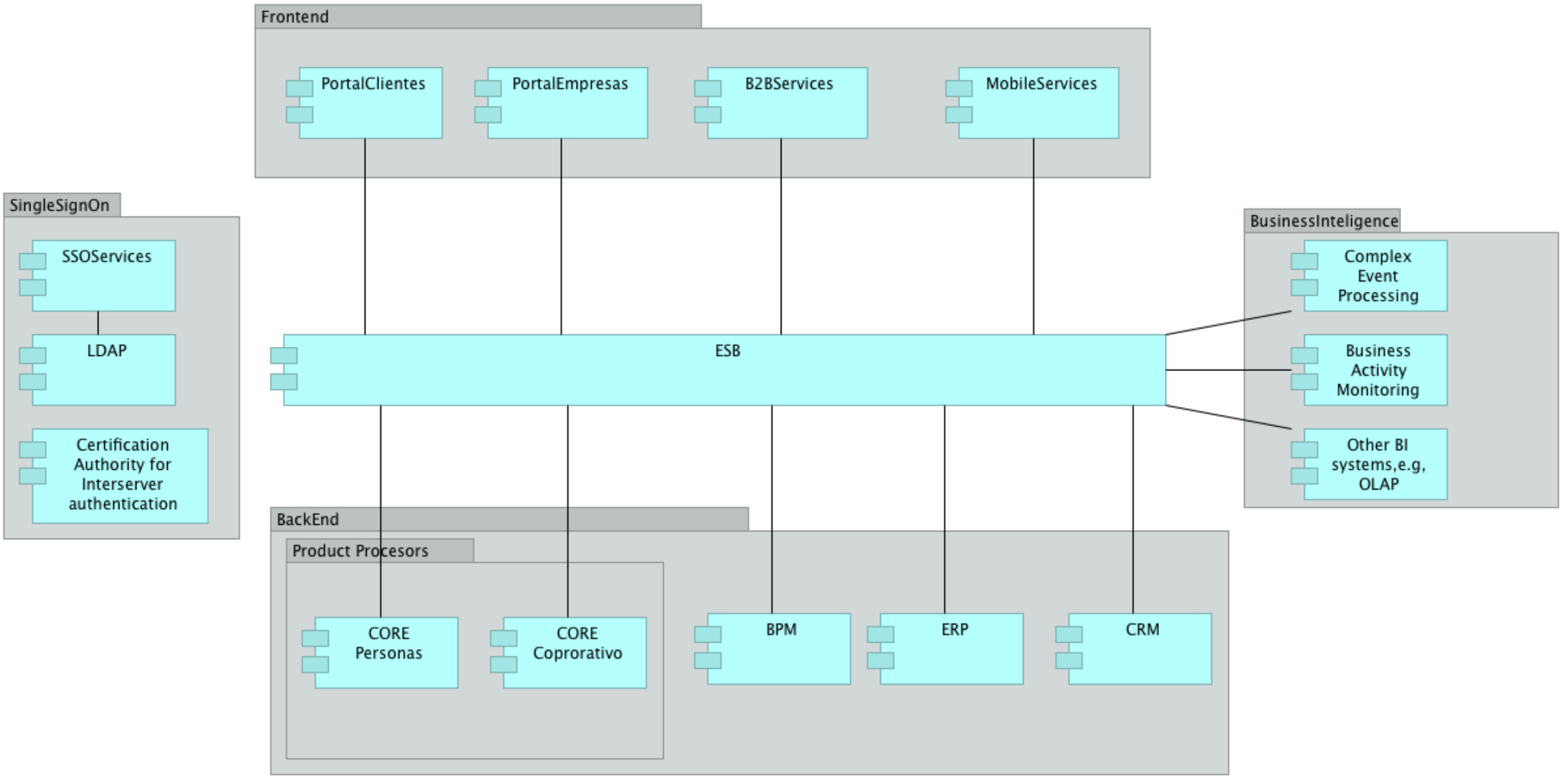
Ejemplos

Ejemplo: The internet Domain Name System



SOA

Patrón que extiende el patrón intermediario



Retos de Integración

Retos fundamentales de la integración

- Las redes no son confiables: demoras e interrupciones, hardware heterogéneo
- Las redes son lentas
- Las aplicaciones son diferentes
- El cambio es inevitable

Cambios en la cultura organizacional

- Negocios organizados de manera funcional
- Áreas de TI organizadas de manera funcional
- Integración empresarial, integra sistemas y unidades de negocio
- ¿Han escuchado algo de Enterprise Architecture?

Implicaciones de alto impacto

- La infraestructura de integración se convierte en punto crítico del sistema
- Aplicar políticas claras de COB
- Riesgo de costo muy alto

Falta de control sobre las aplicaciones participantes

- Aplicaciones empacadas no listas para la integración
- Políticas de control cambios relajadas
- Los EndPoints pueden convertirse en parte funcional de la aplicación base

Muchos/Pocos estándares disponibles

- JMS
- XML
- WebServices: SOAP, WSDL
- Cada empresa vende su propio estándar
- Los estándares disponibles cubren solo una fracción de las necesidades de integración

Operación de una solución EAI

- Monitoreo
- Control de cambios
- COB
- Seguridad
- Manejo de problemas
- Manejo de proyectos
- Seguridad

Bibliografía

- Saltzer, Kashoek. Principles of Computer System Design. Morgan Kaufmann. 2009.

- Hohpe et al. Enterprise Integration Patterns. Adison Wesley. 2004.

- O'Brien,Bass,Merson,. Quality Attributes and service Oriented Architectures. Tecnical Report, Carneige & Mellon SEI. 2005.

- "Service Mediation: The role of an enterprise service Bus" por TIBCO. http://www.tibco.com/multimedia/wp-service-mediation_tcm8-2440.pdf. Visitado por última vez en Junio 2011.

Fin