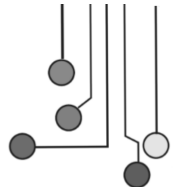


# Acoplamiento e interoperabilidad

Sistemas de Información Orientados a Servicios

RODRIGO SANTAMARÍA

Acoplamiento débil

Tipos de acoplamiento

Cabalgando el acoplamiento

Interoperabilidad: ESB

Conectividad

Mapeado

# Acoplamiento e interoperabilidad

# Acoplamiento



ACOPLAMIENTO DÉBIL

COMUNICACIÓN ASÍNCRONA
TIPOS HETEROGÉNEOS
TIPADO DÉBIL
PATRONES DE INTERACCIÓN
MEDIADORES
COMPENSACIÓN
OTROS MÉTODOS DE DESACOPLAMIENTO

CABALGANDO EL DESACOPLAMIENTO

# Acoplamiento débil



- Objetivo: mejorar los siguientes aspectos
  - Escalabilidad
  - Flexibilidad
  - Tolerancia a fallos
- Medio: minimizar dependencias
  - Menos dependencias → un fallo o modificación en un sistema afectará menos a otros sistemas
- El acoplamiento débil es un principio
  - Uno debe decidir el grado y la forma en que se implementará

# Tolerancia a fallos



- El desarrollo software está expuesto a muchos fallos
  - No es posible entregar programas perfectamente diseñados y robustos, sobre todo con las demandas del mercado
- No obstante, los fallos son costosos
  - En un sistema de reserva de billetes de avión: 100.000 $/h
  - En un sistema de tarjetas de créditos: 300.000 $/h
  - En un sistema de intercambio de acciones: 8.000.000 $/h
- Necesidad de minimizar sus efectos y consecuencias

# Acoplamiento fuerte y débil en SOA



| | Tight coupling | Loose coupling |
|---|---|---|
| **Physical connections** | Point-to-point | Via mediator |
| **Communication style** | Synchronous | Asynchronous |
| **Data model** | Common complex types | Simple common types only |
| **Type system** | Strong | Weak |
| **Interaction pattern** | Navigate through complex object trees | Data-centric, self-contained message |
| **Control of process logic** | Central control | Distributed control |
| **Binding** | Statically | Dynamically |
| **Platform** | Strong platform dependencies | Platform independent |
| **Transactionality** | 2PC (two-phase commit) | Compensation |
| **Deployment** | Simultaneous | At different times |
| **Versioning** | Explicit upgrades | Implicit upgrades |

[Josuttis07]

- En SOA, la mayoría de los aspectos tendrán un acoplamiento débil
  - Aunque no hay una métrica, se espera que sea así

# Comunicación: asíncrona



- El envío de mensajes no implica espera por la respuesta
  - No sabemos cuándo (o si) habrá respuesta
    - Hay que tratar con ella cuando llegue
    - Hay que asociarla al mensaje de envío original
    - Hay que procesar la respuesta teniendo en cuenta el estado y contexto asociado al envío original
- Ventaja: mejora del rendimiento
  - Los sistemas que intercambian el servicio no tienen que estar online al mismo tiempo
- Desventaja: lógica más compleja en el consumidor
  - Sobre todo si hace falta una respuesta y el envío de mensajes es alto

# Tipos de datos: heterogéneos



- En un sistema distribuido de gran escala es muy difícil armonizar los tipos de datos
  - Propietarios diferentes → difícil llegar a acuerdos
  - El sistema puede hacerse muy complicado
  - El sistema nunca se completa → parálisis por análisis
- Si se consigue, es muy costoso mantener los sistemas armonizados
  - Acoplamiento fuerte entre sistemas: un cambio en un atributo de un tipo de datos en un sistema debe reflejarse en todos

# "WHAT A SHAME THAT WE CAN'T HARMONIZE ANYMORE"

A while ago, I gave a talk about SOA that included my usual claim that you have to accept that you can't harmonize data types on large systems. A senior systems architect came to me during the coffee break and said, "Isn't it sad how bad things have become in our industry when we even can't harmonize data types anymore? In the past we were able to analyze and design our systems." It sounded like a criticism of the younger generations of software developers, who are able to do only one thing: copy and paste.

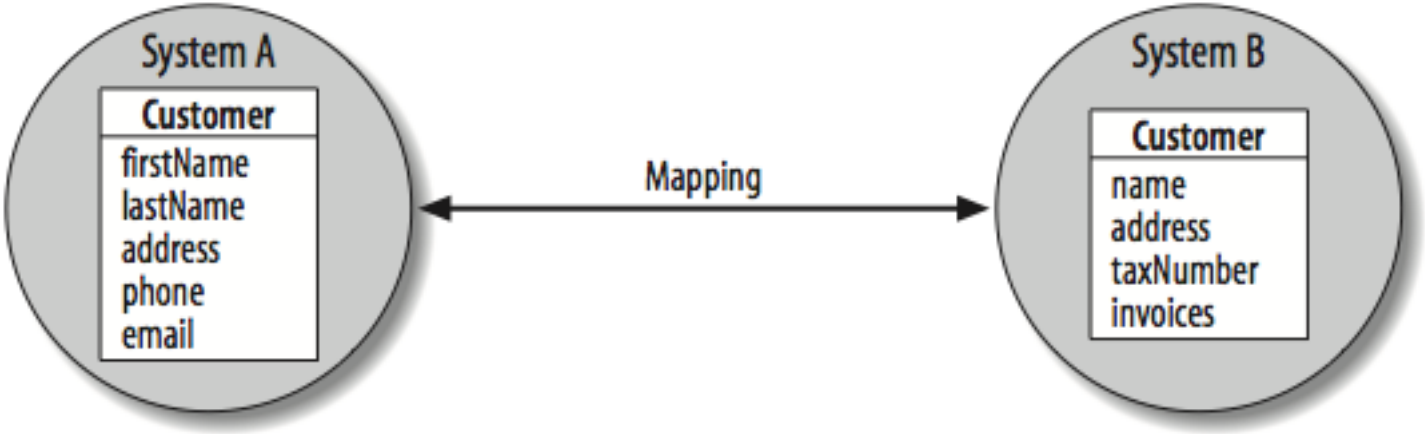
My answer was as follows: "You might be right. In our crazy times, we simply do not have enough time for careful and long analysis and design. The marketing guys rule the world, and if we don't deliver in time, we are out of the market (even if we have better quality). But be careful, and don't underestimate the level of complexity we have reached now by connecting systems. Remember, there is no central control any longer, and if there were it wouldn't work. If we had harmonized an address type before we shipped the Internet protocols, the Internet would never have become reality. Large-scale systems need the minimal consensus you can provide to be successful. Note that you still can harmonize data types when things run."

[Josuttis07]

# Tipos de datos heterogéneos



- Un dato tiene distintos tipos en distintos sistemas




- Se necesita un *mapeado* que traduzca tipos
  - Añade complejidad
  - Mantiene desacoplados los sistemas

# Tipado débil



- A priori, el tipado fuerte es mejor porque detecta errores en tiempo de compilación en vez de en tiempo de ejecución
  - Pero si un sistema crece, debemos tener en cuenta que el tipado fuerte conlleva tiempo y requiere información
    - P. ej. si nuestros datos tienen tipado fuerte, la infraestructura de comunicación (ESB) debe sincronizar los cambios en los datos con los sistemas que los generan/aceptan. Si Internet tuviera que hacer esto, jamás habría podido escalar
- De nuevo, qué grado de tipado tenemos depende del problema y de los requisitos

# Patrones de interacción



- Tipos de datos que se consideran fundamentales y de qué manera se combinan:
  - Definir datos fundamentales (cadenas normalmente)
    - Pero se pueden añadir otros: enteros, reales, booleanos, etc.
  - Limitar valores posibles (p. ej. cierto formato de cadena)
  - Definir la posibilidad de tipos superiores (estructuras, objetos)
  - Definir la posibilidad de tipos secuenciales (arrays, listas, etc.)
  - Definir las relaciones entre tipos (herencia, polimorfismo, etc.)
- A mayor riqueza de datos
  - Programación a más alto nivel
  - Más problemas con el mapeado

# Conexión



- Conexión ‘punto a punto’
  - El remitente manda una petición a un sistema usando su dirección física
    - Similar a mandar una carta a una dirección postal
      - ¿Qué pasa si el receptor cambia de casa?
      - ¿Qué pasa si el receptor no está en casa, o si la carta se pierde entre todo el correo comercial?
  - Necesidad de un intermediario → mediador

# Conexión: mediadores (tipos)



- Broker, servidor de nombres
  - La petición se hace mediante un nombre simbólico y el mediador obtiene la dirección física correcta *antes* del envío
    - Los servicios web se pueden ver como conexiones punto a punto con un broker (DNS) para resolver direcciones
- Middleware, ESB
  - La petición se hace mediante un nombre simbólico y el mediador direcciona la petición mediante reglas de enrutado *después* del envío

# Consistencia



- Típicamente, la consistencia en replicación se solventa con un *Commit de Dos Fases* (2PC)
  - Algoritmo de acuerdo distribuido para cometer una transacción de manera atómica
- Fases
  1. **Voto**: un proceso *coordinador* prepara a todos los procesos *participantes* para cometer o abortar la transacción y *votar*
  2. **Cometer**: el *coordinador* decide cometer (si todos los votos son sí) o abortar, y notifica el resultado a los *participantes*
     - Los participantes llevarán a cabo las operaciones necesarias

# Consistencia: compensación



- Mecanismo de acuerdo de filosofía inversa a 2PC
- La transacción se comete en todos los backends
  - Si uno falla, se busca un modo de ‘compensar’ la inconsistencia
    - Deshacer la transacción en los backends que cometieron con éxito
    - Mandar un error para que un operador lo resuelva manualmente
- Análisis
  - Ventaja: la transacción no es síncrona → menor acoplamiento
  - Desventaja: se requiere una invocación explícita de servicios que reviertan la transacción o resuelvan manualmente el error

# Otros métodos de desacoplamiento



- Control de la lógica de proceso
  - Descentralización para desacoplar y evitar cuellos de botella
- Desarrollo
  - Permitir actualizaciones de los sistemas en distintos momentos
- Dependencias de plataforma
  - El uso de soluciones independientes de plataforma contribuye al desacoplamiento (p. ej. Docker)

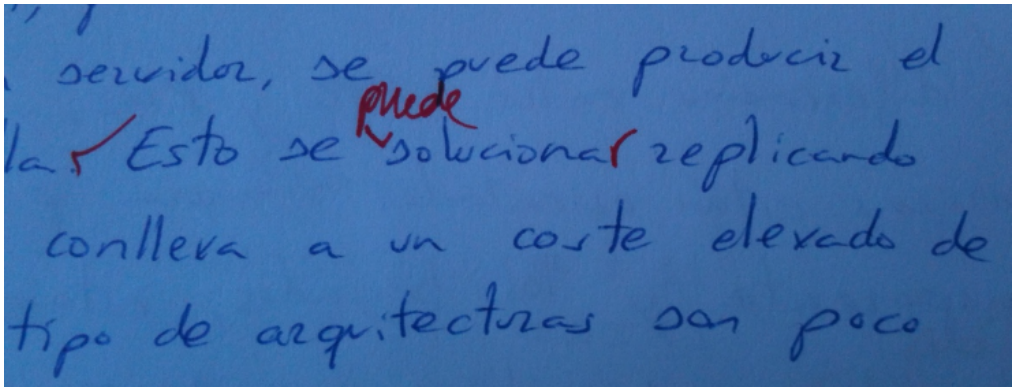
# Cabalgando el desacoplamiento

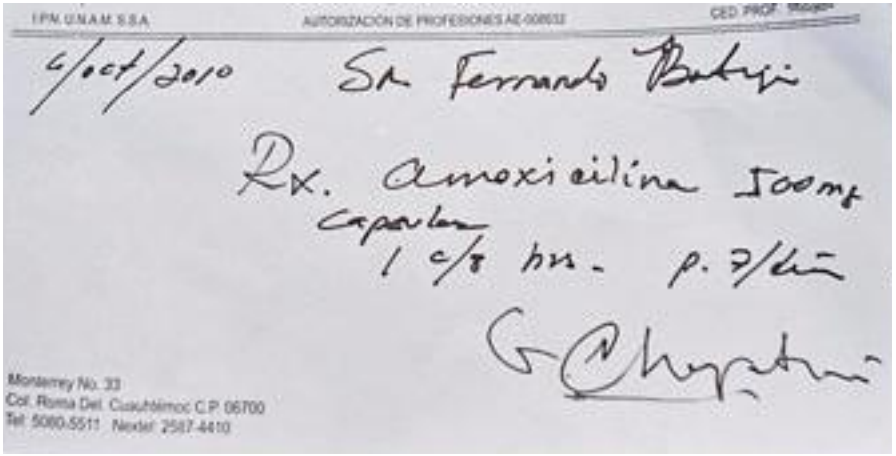


- Dentro de cada contexto y problema, debemos ser capaces de decidir los aspectos y el grado de desacoplamiento que requerimos en el sistema

mi mamá me mima

yo mimo a mi mamá



# Interoperabilidad



ENTERPRISE SERVICE BUS

MAPEADO
CONECTIVIDAD
OTROS VALORES

# Enterprise Service Bus (ESB)

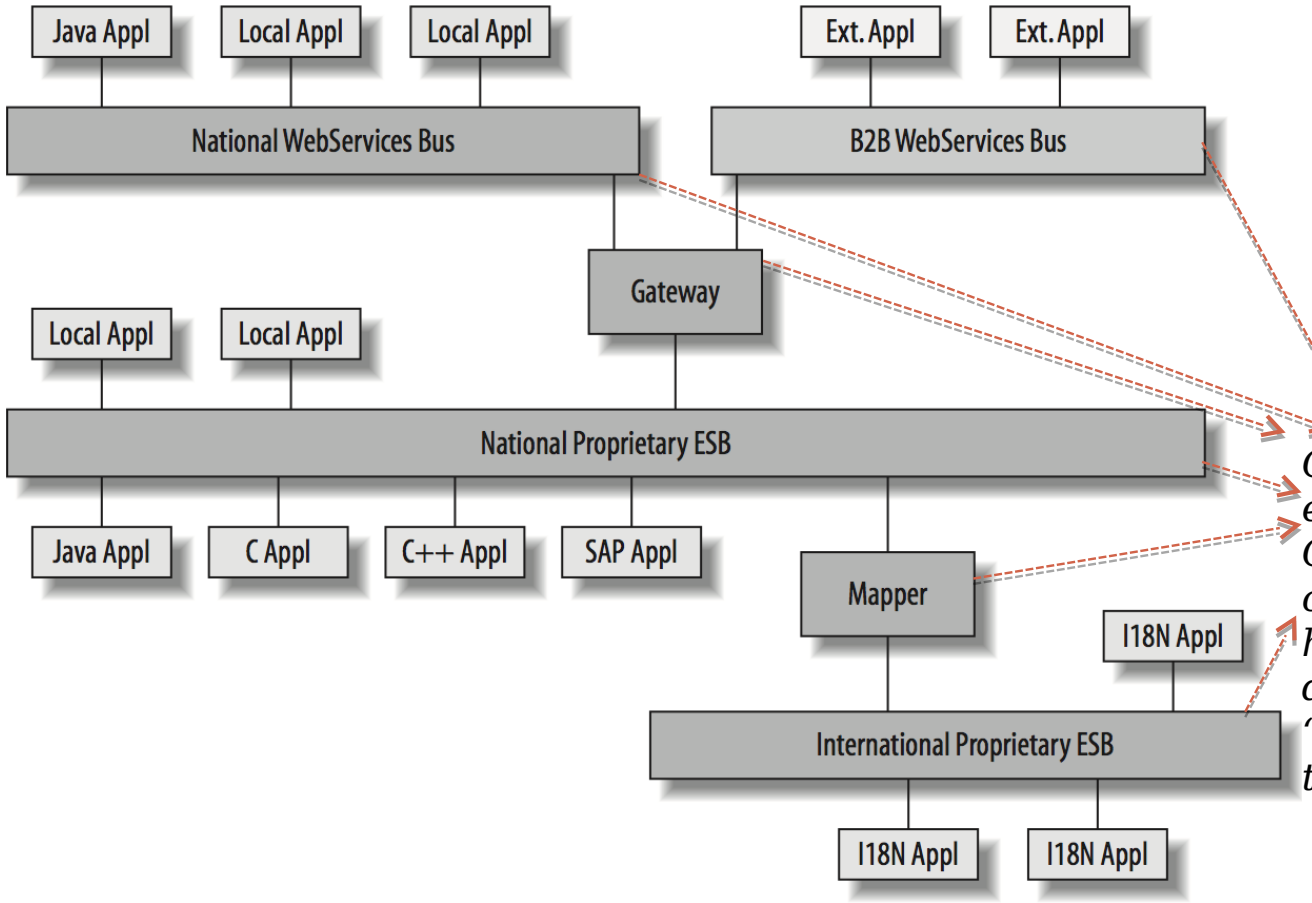


- Infraestructura SOA que se encarga de la invocación a los servicios que suministran los proveedores
- Responsabilidades:
  - Conectividad y direccionamiento
  - Transformación y mapeo de datos
  - Seguridad y fiabilidad
  - Monitorización y registro de datos
- La principal tarea es proveer **interoperabilidad**
  - Es decir,  permitir operar de manera conjunta a componentes con acoplamiento débil

# Heterogeneidad en ESB



- Variabilidad en middleware, protocolos o s. web



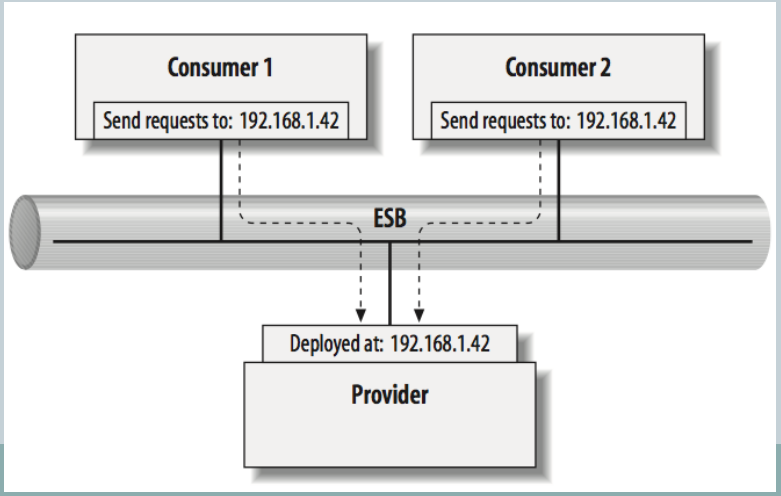

*Conceptualmente, todo es un único ESB. Cualquier sistema puede conectarse con otro. El hecho de tratar con distintos protocolos es 'sólo' una cuestión técnica*

# Conectividad



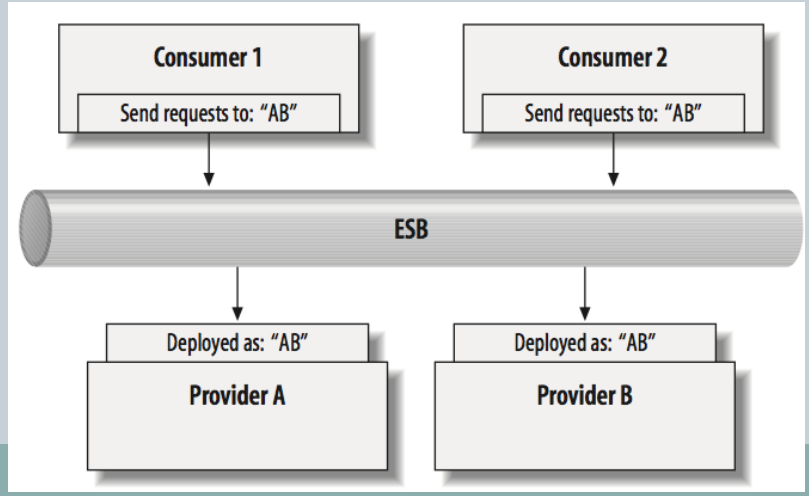
## Punto a punto

- El consumidor debe conocer la dirección exacta (*endpoint*) del proveedor
  - Puede fallar si el proveedor no está disponible




## Mediación

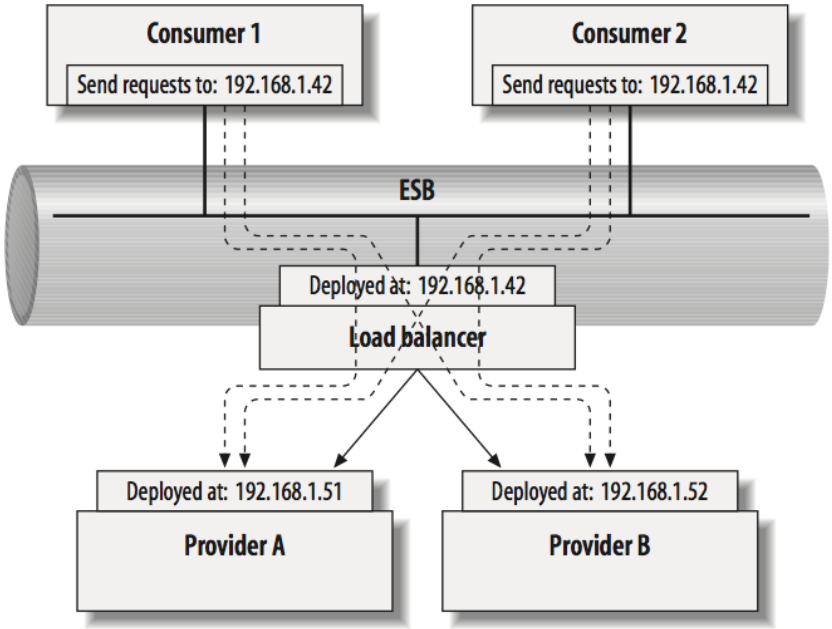
- ESB resuelve el *endpoint* a partir de una etiqueta
  - Permite lidiar con modificaciones dinámicas: balanceo de carga, desconexiones, etc



# Interceptores



- Similar al mediador, pero el proveedor no suministra ‘etiquetas’
  - Reemplaza la dirección física a la que envía el consumidor por otra según criterios de equilibrio de carga, caídas, etc.



# Conexión y servicios web



- Un servicio web es punto a punto por definición
  - Intercambiar servicios implica conocer el punto final del servicio
  - No tiene en cuenta el equilibrio de carga o las caídas
    - Tarde o temprano es necesario incorporar a un ESB basado en servicios web un interceptor o mediador

# Mapeado: protocolos y APIs



- Un ESB puede basarse en protocolos o en APIs
- Protocolos
  - Los participantes se envían mensajes según un protocolo definido por el ESB y al que los participantes deben ajustarse
    - P. ej. SOAP
- API
  - ESB provee una API específica para cada plataforma, que utilizan los participantes para implementar y llamar servicios
    - ESB se encargará del mapeo al protocolo de comunicación
    - P. ej. interfaces Java

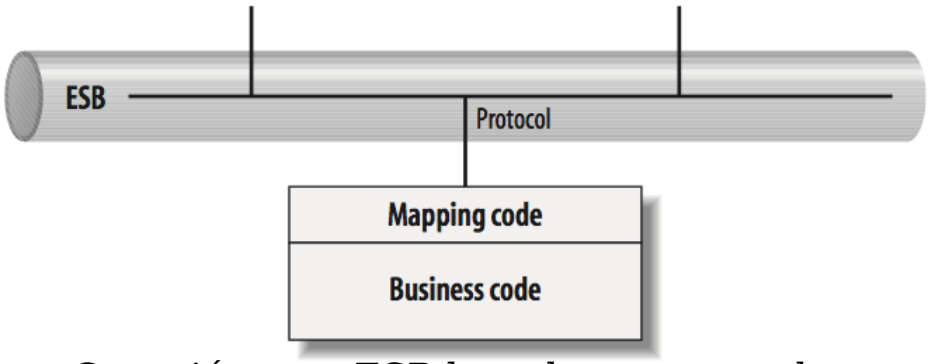
# Mapeado y desarrollo



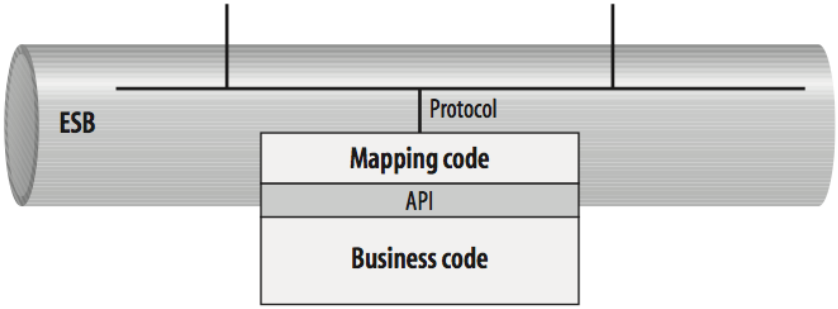
- Basarse en protocolos da más responsabilidad a los desarrolladores y consumidores de servicios
  - ↑ Desacopla la infraestructura del desarrollo de servicios
  - ↓ Cada desarrollador va a tener que solucionar el mismo problema
- Suele derivar en una tercera capa de comunicación entre el servicio y el protocolo
  - Ejemplo: Internet
    - Capa de protocolo: HTTP(S)
    - Capa de servicio: Página web/acceso programático
    - → Capa de comunicación: Navegadores/APIs

# Capas de negocio a protocolo

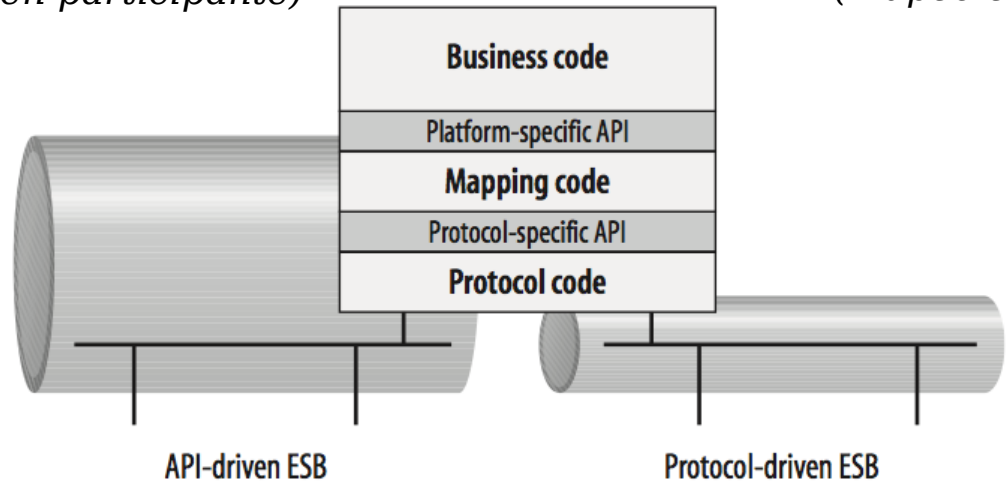





*Conexión a un ESB basado en protocolo (mapeo en participante)*




*Conexión a un ESB basado en API (mapeo en ESB)*




*Resultado final frecuente: el participante se adhiere a una API pero internamente también hay un mapeo al protocolo estable*

# Otros valores añadidos de ESB



- Mapeado de datos
  - Según las responsabilidades de ESB, puede encargarse del mapeado de tipos o no, o mantener una capa fina de mapeado para direccionamiento
- Direccionamiento inteligente
  - Diferentes prioridades según proveedor/consumidor
  - Diferente tratamiento según contenido
- Seguridad y fiabilidad → en temas posteriores
- Monitorización y registro → en temas posteriores

# API Gateway



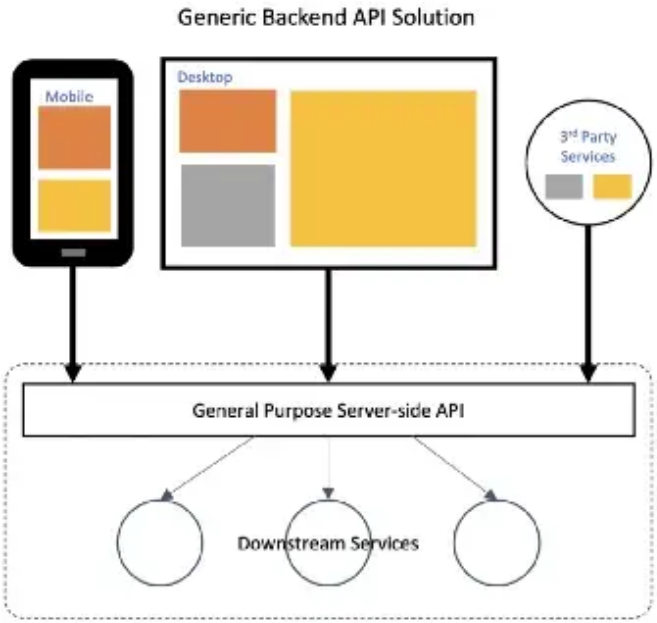
- ESB que se encarga de:
  - Dividir una petición remota en las distintas invocaciones de servicios necesarios
  - Orquestar la respuesta
  - Realizar el mapeado de protocolos necesario
  - Adicionalmente:
    - Tareas de autenticación y seguridad
    - Tareas de rendimiento (balanceo de carga)
    - Cumplimiento de SLAs
    - Tareas de monitorización
    - Acceso como servicios a aplicaciones antiguas (legacy)

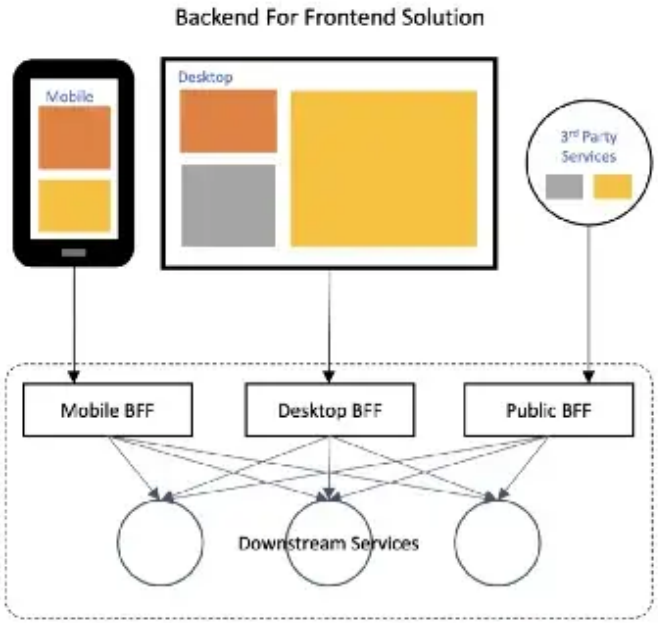
Más información: IBM. *What are API Gateways?* **2022** (enlace)

# Backend For Frontent (BFF)



- Variante de la API Gateway que ofrece una API (backend) adaptada a distintos clientes (frontends)
  - Muy usada para diferenciar frontend móviles y de escritorio




General-purpose API vs BFF, imagen de Szczepanik, M. “Backend for frontend (BFF) pattern -- why do you need to know it?” Revista *Medium*, **2021** (enlace)

# Ejercicio



- Tenemos la siguiente situación (ejemplo real)
  - Una compañía usaba como ESB la tecnología propietaria de un proveedor EAI (Enterprise Application Integration)
    - Lo llamaremos “ESB propietario nacional”
    - A él se conectan aplicaciones en C, C++ y Java
  - Luego se fusionó con otra compañía de otro país, que usaba la misma tecnología middleware pero que no garantizaba la interoperabilidad
    - Lo llamaremos “ESB propietario internacional”
    - Se conectan tres aplicaciones de internacionalización (i18n)

# Ejercicio



- El sistema continuó creciendo
  - Los servicios web se convierten en un estándar de facto, y se necesita ampliar la interoperabilidad a estos servicios
  - Por razones de seguridad, queremos dos buses de servicios web:
    - Uso interno con tres aplicaciones locales
    - Uso B2B (business to business), con dos aplicaciones externas
- ¿Qué aspecto crees que tiene el ESB de esta empresa tras sufrir todos estos cambios a lo largo del tiempo?

# Resumen



- El **acoplamiento débil** es un concepto fundamental de SOA que busca reducir dependencias entre sistemas
- Hay varias formas de acoplamiento débil: uno debe encontrar la **combinación apropiada** para cada proyecto y contexto
- Toda forma de acoplamiento débil tiene **desventajas** además de ventajas, así que nunca debe ser un fin en sí mismo
- La necesidad de **mapear** datos es, normalmente, una cualidad **positiva** en sistemas grandes

- **ESB** es la **infraestructura** de SOA, proveyendo de interoperabilidad (conectividad, mapeo y direccionamiento)
- Un ESB suele ser **heterogéneo**
- Decidir si nuestro ESB está basado en **API** o en **protocolo** es fundamental
  - Si ESB define un protocolo, es responsabilidad de los participantes adaptarse a él. Desacoplamiento ESB/participante pero cualquier cambio en el protocolo afecta a todos los participantes
  - Si ESB provee una API para los participantes, hace transparentes los detalles del protocolo, pero requiere que el ESB genere ese código para los clientes

# Referencias




- [Josuttis07] Nicolai M. Josuttis. *SOA in practice. The Art of Distributed System Design*. O'Reilly, **2007**. Ch 4/5.