



Universidad de Oviedo



Integración



Curso 2019/20

Jose Emilio Labra Gayo

Integración

Integración de aplicaciones
Gran reto de la informática



Integración

Estilos de integración

Transferencia de ficheros

Base de datos compartida

Invocación Procedimiento Remotos

Mensajería

Event log

Topologías

Hub & Spoke, Bus

Arquitecturas orientadas a servicios

WS-*, REST

Microservicios

Serverless

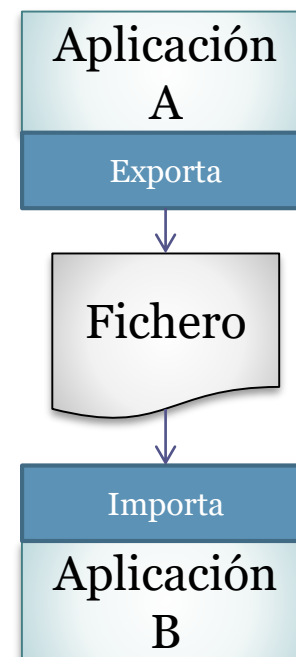
Estilos de integración

- Transferencia de ficheros
- Base de datos compartida
- Invocación procedimiento remoto
- Mensajería
- Event log

Transferencia de ficheros

Una aplicación genera un fichero de datos que es consumido por otra

Una de las soluciones más comunes



Transferencia de ficheros

Ventajas

Bajo acoplamiento

Independencia entre
aplicación A y B

Facilita depuración

Se pueden analizar datos
del fichero

Problemas

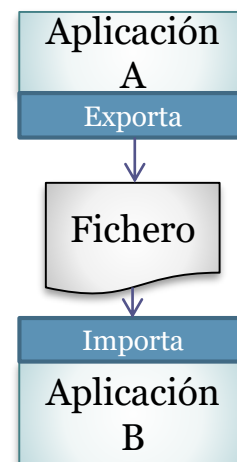
Acordar formato de fichero común

Puede aumentar acoplamiento

Coordinación

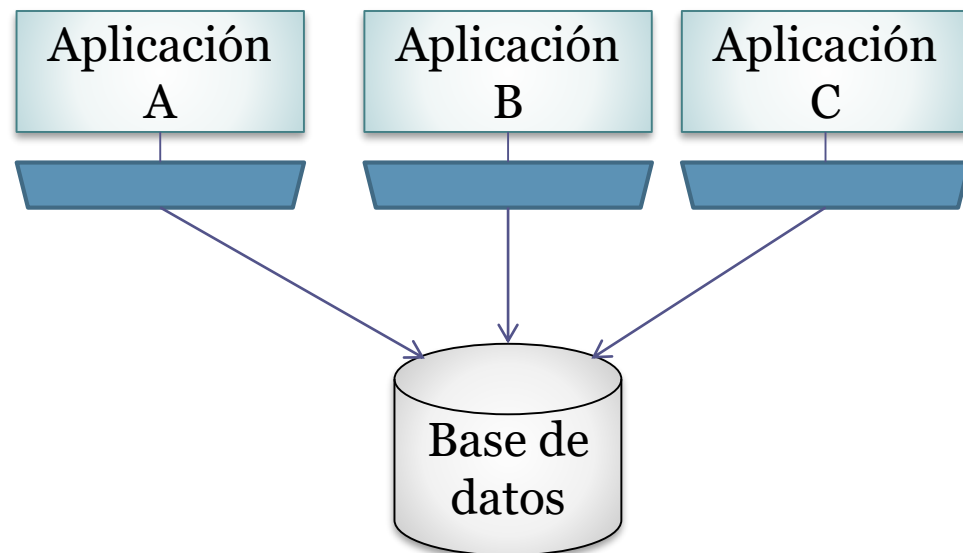
Una vez enviado el fichero, la
aplicación B puede modificarlo ⇒
¡2 ficheros!

Suele requerir intervención manual



Base de datos compartida

Las aplicaciones almacenan sus datos en una base de datos común



Base de datos compartida

Ventajas

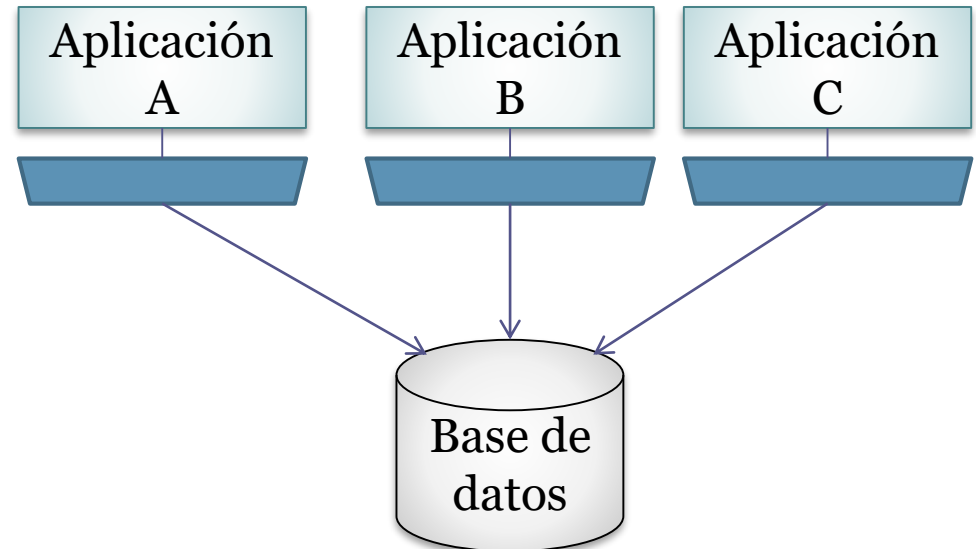
Datos siempre disponibles

Todo el mundo accede a la misma información

Consistencia

Formato familiar

SQL para todo?



Base de datos compartida

Problemas

El esquema de la base de datos puede variar

Requiere esquema común para todas las aplicaciones

Fuente de problemas y conflictos

Necesidad de paquetes externos (acceso BD común)

Rendimiento y escalabilidad

Base de datos como cuello de botella

Sincronización

Problema con bases de datos distribuidas

Escalabilidad

NoSQL ?

Base de datos compartida

Variaciones

Data warehousing: Base de datos utilizada para análisis de datos e informes

ETL: proceso basado en tres fases

Extracción: Obtención de fuentes heterogéneas

Transformación: Procesado de los datos

Carga (Load): Almacenamiento en base de datos compartida

Invocación procedimiento remoto

Una aplicación invoca una función de otra aplicación que puede estar en otra máquina

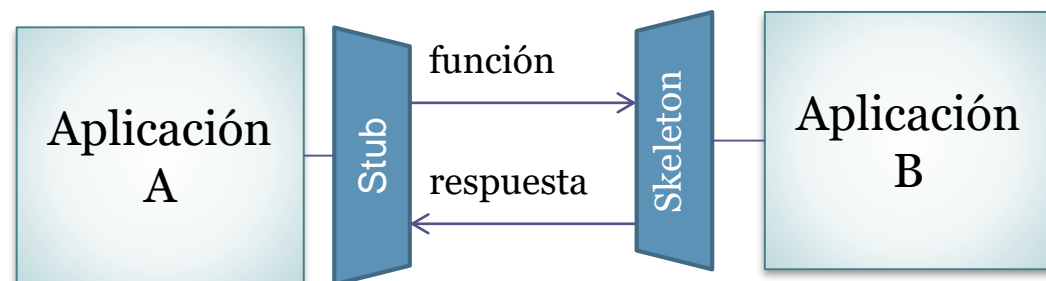
En la invocación puede pasar parámetros

Obtiene una respuesta

Gran variedad de aplicaciones

RPC, RMI, CORBA, .Net Remoting, ...

Servicios web, ...



Invocación procedimiento remoto

Ventajas

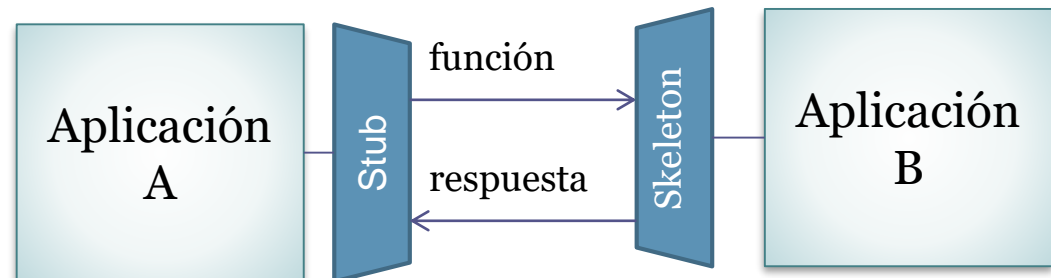
Encapsulación de implementación

Múltiples interfaces para la misma información

Se pueden ofrecer distintas representaciones

Modelo familiar para desarrolladores

Similar a llamar a un método



Invocación procedimiento remoto

Problemas

Falsa sensación de sencillez

Procedimiento remoto \neq Procedimiento

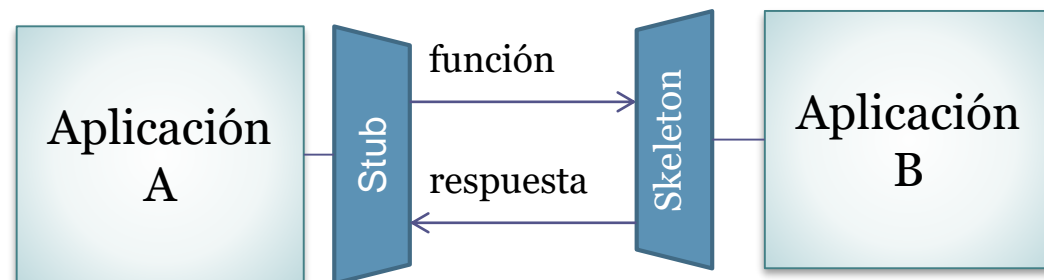
8 falacias de computación distribuida

Invocaciones mediante sincronización

Aumenta acoplamiento entre aplicaciones

La red es fiable
La latencia es cero
El ancho de banda es infinito
La red es segura
La topología no cambia
Hay un administrador
El coste de transporte es cero
La red es homogénea

8 falacias computación distribuida



Invocación procedimiento remoto

Nuevas revisiones: gRPC

Propuesta de Google (<https://grpc.io/>)

Marco de aplicaciones de mensajería de alto rendimiento

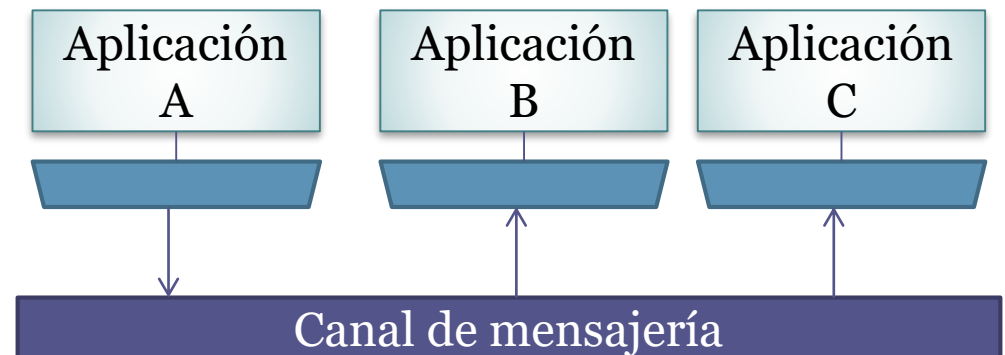
Basado en http/2

Mensajería

Múltiples aplicaciones independientes se comunican enviando mensajes a un canal

Comunicación asíncrona

Las aplicaciones envían mensajes y continúan ejecutándose



Mensajería

Ventajas

Bajo acoplamiento

Aplicaciones independientes entre sí

Comunicación asíncrona

Las aplicaciones continúan la ejecución

Encapsulación

Sólo se expone el tipo de mensajes

Problemas

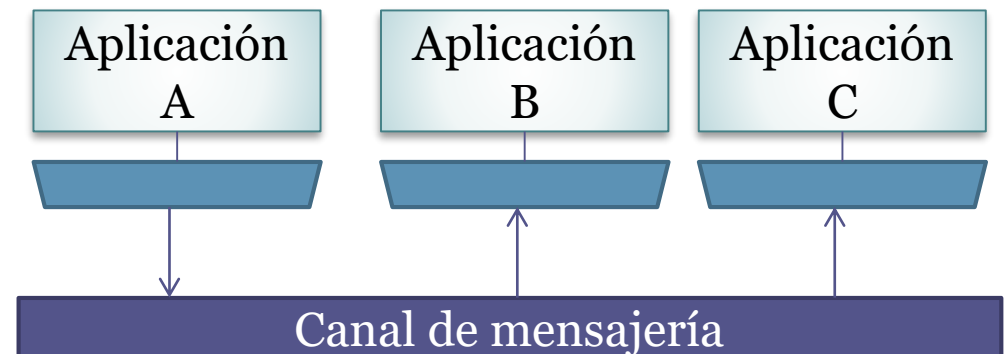
Complejidad de implementación

Comunicación asíncrona

Transformación de datos

Adaptación formato de mensajes

Diferentes topologías



Topologías de integración

Hub & Spoke

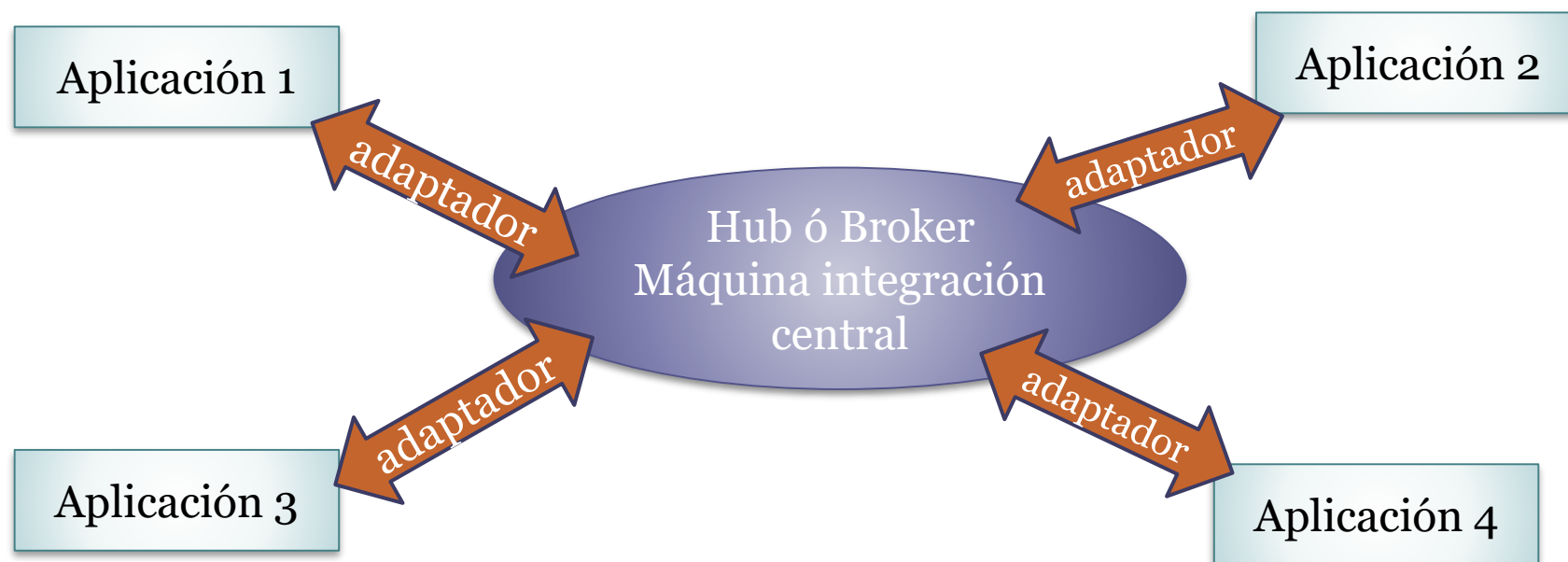
Bus

Hub & Spoke (central/radial)

Relacionado con patrón Bróker

Hub = Bróker centralizado de mensajes

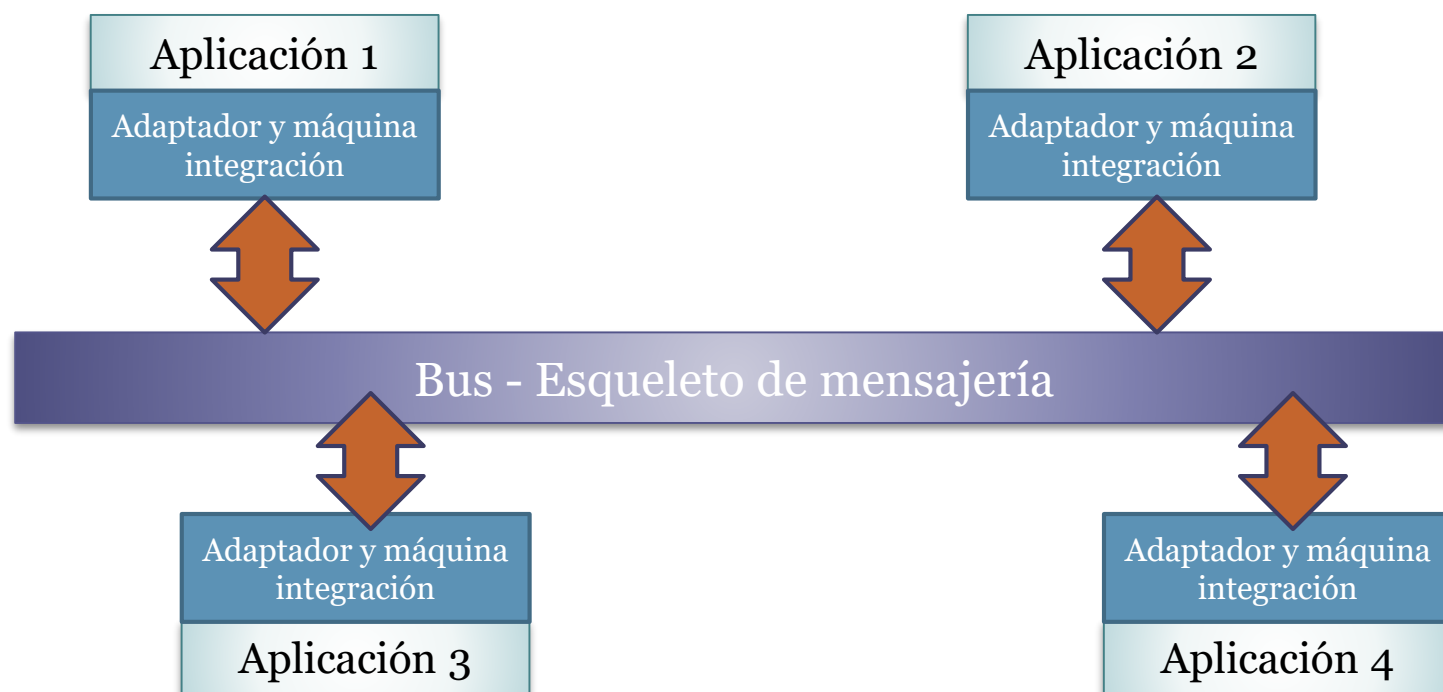
Se encarga de la integración



Bus

Cada aplicación contiene su máquina de integración

Estilo Publish/Subscribe



Bus

ESB - Enterprise Service Bus

Define un esqueleto (*backbone*) de mensajería

Conversión de protocolos

Transformación de formatos

Enrutamiento

Proporciona un API para desarrollar servicios

MOM (Message Oriented Middleware)

Servicios

SOA - Service Oriented Architectures

WS-*

REST

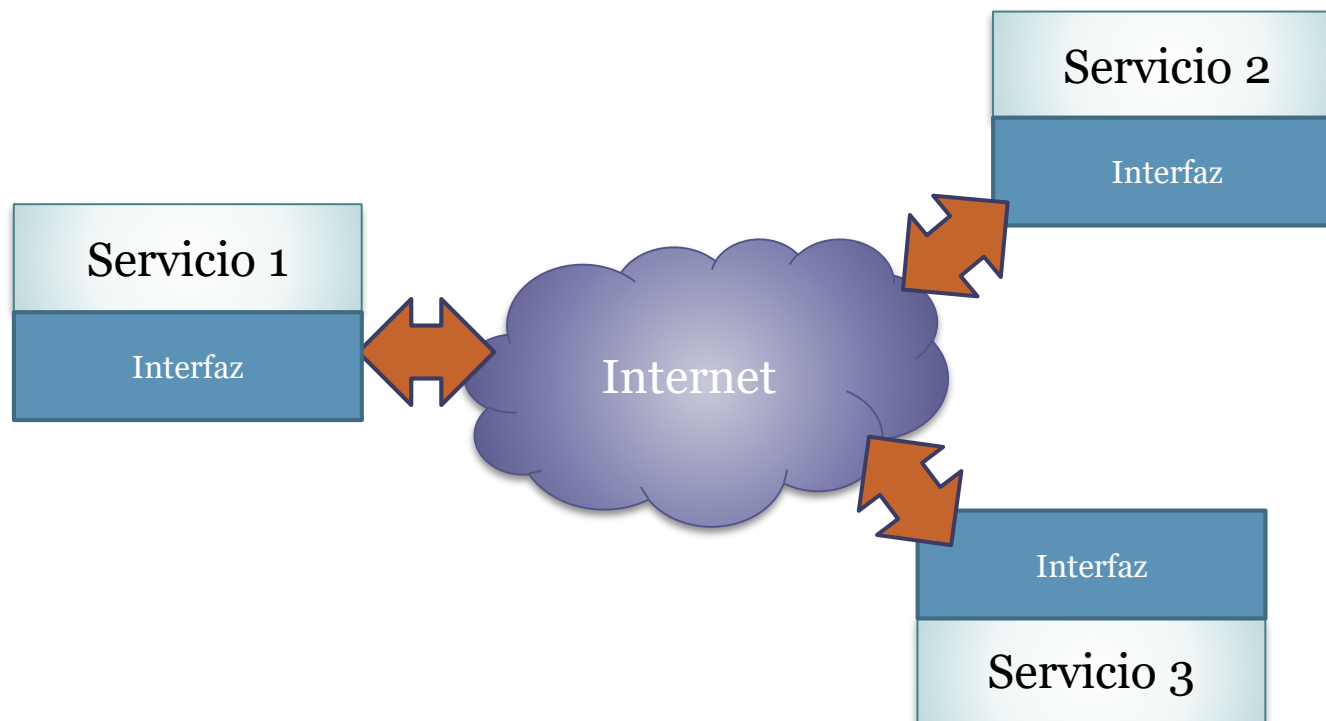
Microservicios

Serverless

SOA

SOA = Service Oriented Architecture

Los servicios están definidos mediante un interfaz



SOA

Elementos

Proveedor : Proporciona el servicio

Consumidor: Realiza peticiones al servicio

Mensajes: Información intercambiada

Contrato o interfaz: Descripción de la funcionalidad ofrecida por el servicio

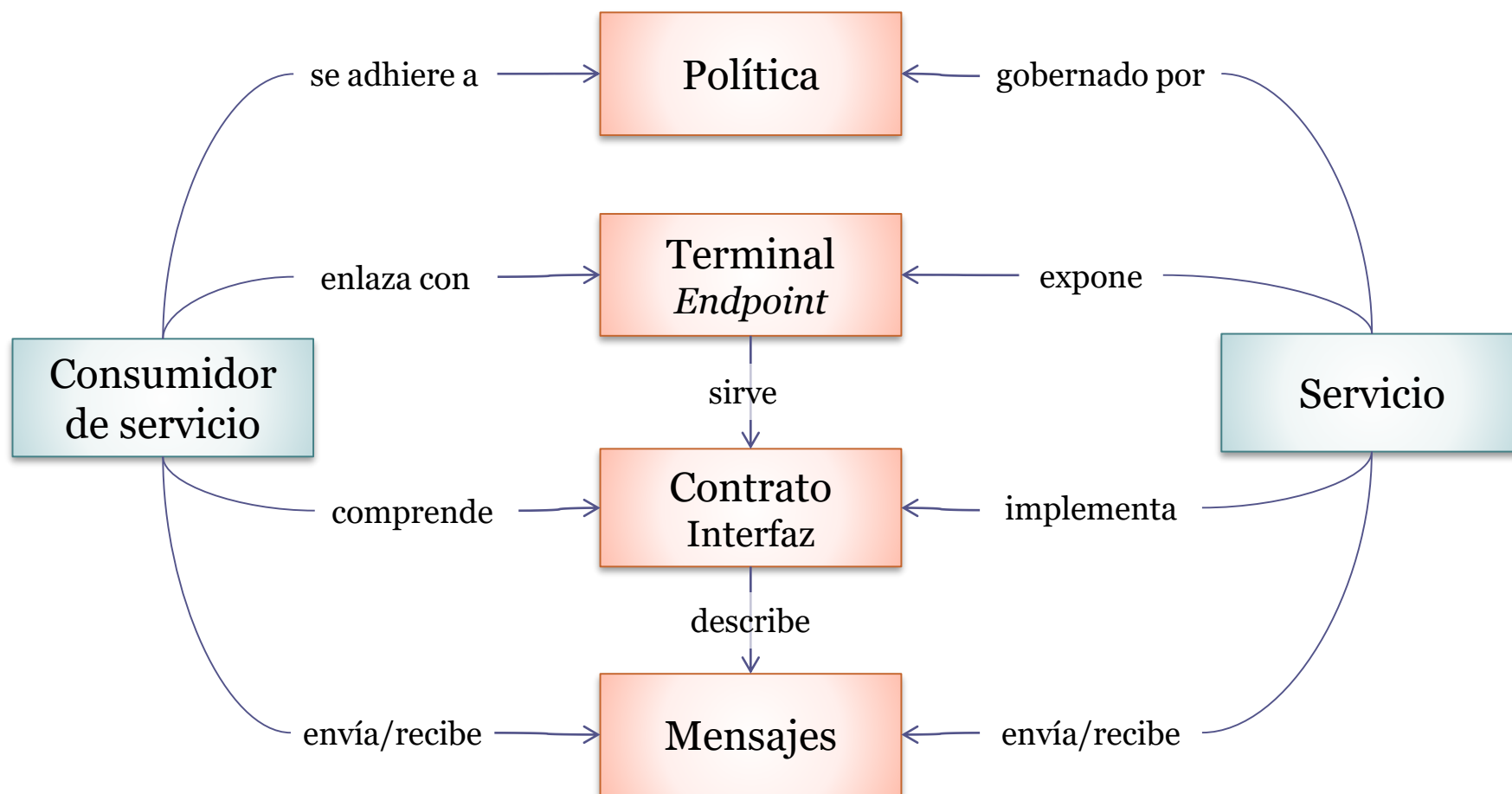
Terminal: Ubicación del servicio

Política: Acuerdos de gobierno del servicio

Seguridad, rendimiento, etc.

SOA

Restricciones



SOA

Ventajas

Independencia de lenguaje y
plataforma

Interoperabilidad

Utilización de estándares

Acoplamiento débil

Descentralizado

Reusabilidad

Escalabilidad

Comunicación uno-a-uno
frente uno-a-muchos

Mantenimiento sistemas

legacy

Añadir una capa de servicios
web

Problemas

Eficiencia.

Puede no ser necesario en:
Entornos muy homogéneos,
tiempo real, ...

Exposición abierta

Riesgo de exponer API al
exterior

Seguridad

Composición de servicios

SOA

Variantes:

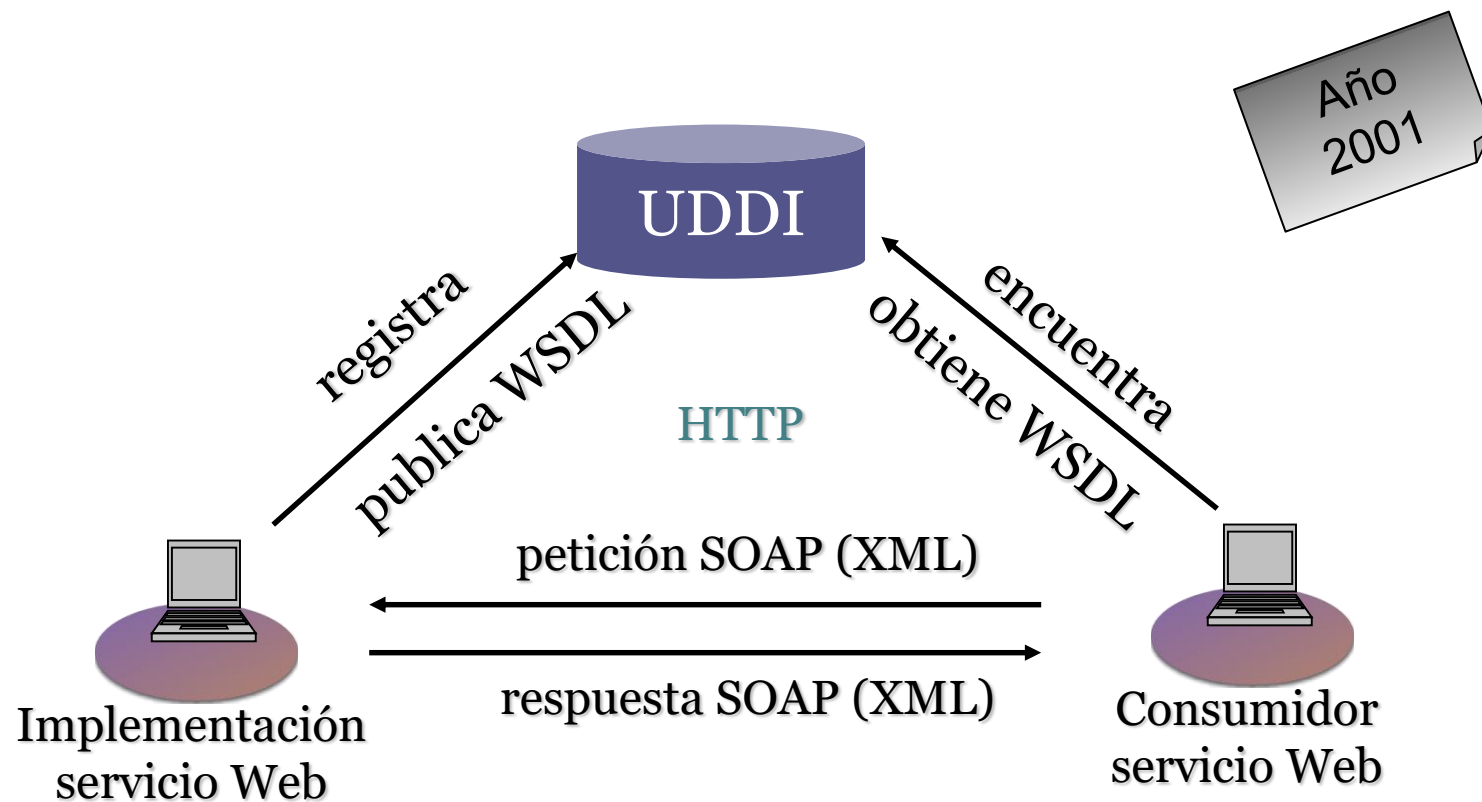
WS-*

REST

WS-*

Modelo WS-* = Conjunto de especificaciones
SOAP, WSDL, UDDI, etc....
Propuesto por W3c, OASIS, WS-I, etc.
Objetivo: Implementación SOA de referencia

WS-*



WS-*

Ecosistema de servicios Web

Año
2001

WS-*

SOAP

Define el formato de los mensajes y varios enlaces con protocolos

Originalmente *Simple Object Access Protocol*

Evolución

Desarrollado a partir de XML-RPC

SOAP 1.0 (1999), 1.1 (2000), 1.2 (2007)

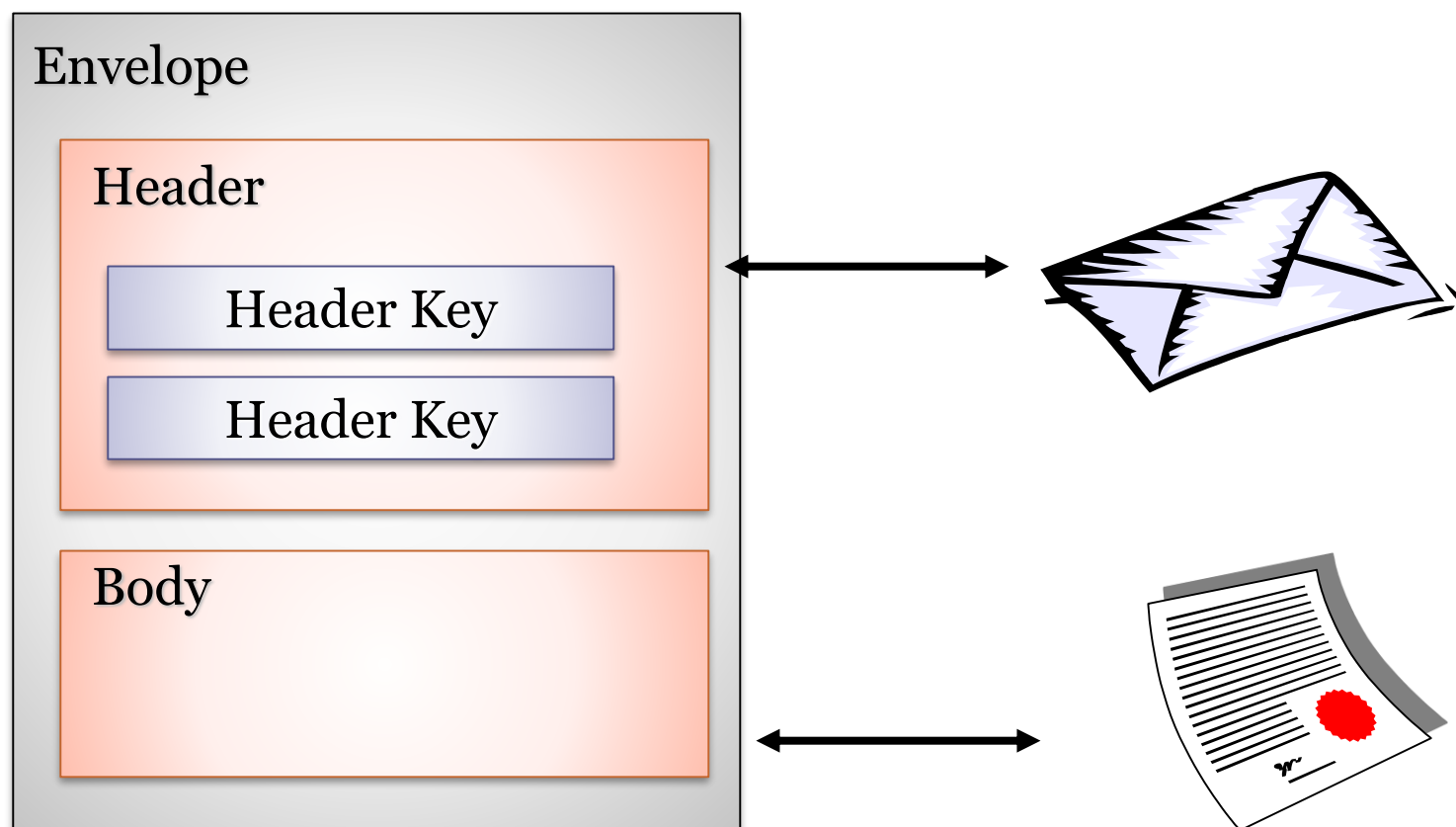
Participación inicial de Microsoft

Adopción posterior de IBM, Sun, etc.

Bastante aceptación industrial

WS-*

Esquema de SOAP



WS-*

Ejemplo de SOAP sobre HTTP

Año
2001

POST ?

```
POST /Suma/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: longitud del mensaje
SOAPAction: "http://tempuri.org/suma"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <suma xmlns="http://tempuri.org/">
      <a>3</a>
      <b>2</b>
    </suma>
  </soap:Body>
</soap:Envelope>
```

WS-*

Ventajas

Especificaciones
realizadas por
comunidad

W3c, OASIS, etc.

Adopción industrial

Implementaciones

Visión integral de
servicios web

Numerosas extensiones:

Seguridad, orquestación,
coreografía, etc.

Problemas

No todas las
especificaciones están
maduras

Sobre-especificación

Falta de implementaciones

Abuso del estilo RPC

Interfaz no uniforme

No se sigue arquitectura
HTTP

Métodos GET/POST
sobrecargados

WS-*

SOAP en la práctica

Numerosas aplicaciones utilizan SOAP

Ejemplo: eBay (50mill. transacciones SOAP al día)

Pero...algunos servicios web populares dejaron de ofrecer soporte SOAP

Ejemplos: Amazon, Google, etc.

REST

REST = REpresentational State Transfer

Estilo de arquitectura

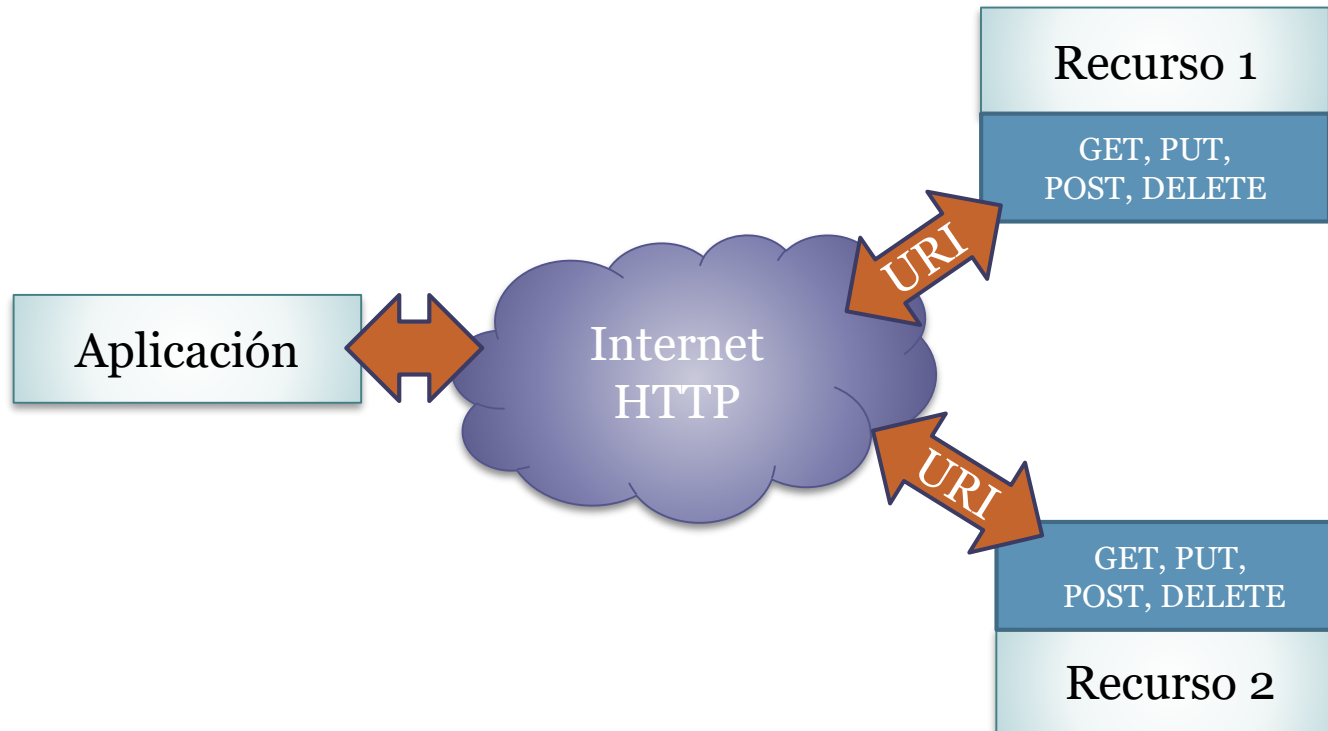
Origen: Tesis doctoral de Roy T Fielding (2000)

Inspirado en la arquitectura de la Web (HTTP/1.1)



REST

REST - Representational State Transfer
Transferencia de representación de estado



REST

Conjunto de restricciones

- Recursos con interfaz uniforme

 - Identificables mediante URIs

- Se devuelven representaciones de los recursos

- Sin estado

- Interfaces genéricos

 - Conjunto acciones: GET, PUT, POST, DELETE

REST = estilo de arquitectura

- Varios niveles de adopción:

 - RESTful

 - Híbrido REST-RPC

REST

En capas

Cliente-servidor

Sin estado

Caché

Servidor replicado

Interfaz uniforme

Identificación de recursos (URIs)

Manipulación de representaciones de recursos

Mensajes auto-descriptivos (tipos MIME)

Enlaces a otros recursos (HATEOAS)

Código bajo demanda (opcional)

REST

Interfaz uniforme:

Conjunto de operaciones limitado

GET, PUT, POST, DELETE

Conjunto limitado de tipos de contenidos

En HTTP se identifican mediante tipos MIME: XML , HTML...

Método	En Bases de datos	Función	Segura?	Idempotente?
PUT	≈Create/Update	Crear/actualizar	No	Si
POST	≈Update	Crea/actualiza subordinado	No	No
GET	Retrieve	Consultar recurso	Si	Si
DELETE	Delete	Eliminar recurso	No	Si

Seguro = No modifica los datos del servidor

Idempotente = El efecto de ejecutarlo n-veces es el mismo que el de ejecutarlo 1 vez

REST

Protocolo cliente/servidor sin estado

Estado gestionado por el cliente

HATEOAS (*Hypermedia As The Engine of Application State*)

Respuestas devuelven URIs a opciones disponibles

Peticiones sucesivas de recursos

Ejemplo: Gestión de alumnos

1.- Obtener lista de alumnos

GET `http://ejemplo.com/alumnos`

Devuelve lista de URIs de alumnos

2.- Obtener información de ese alumno

GET `http://ejemplo.com/alumnos/id2324`

3.- Actualizar información de ese alumno

PUT `http://ejemplo.com/alumnos/id2324`

REST

Ventajas

Cliente/Servidor

- Separación de responsabilidades

- Acoplamiento débil

Interfaz uniforme

- Facilita comprensión

- Desarrollo independiente

Escalabilidad

- Mejora tiempos de respuesta

Menor carga en red (caché)

- Ahorro de ancho de banda

Problemas

REST mal entendido

- Uso de JSON o XML sin más

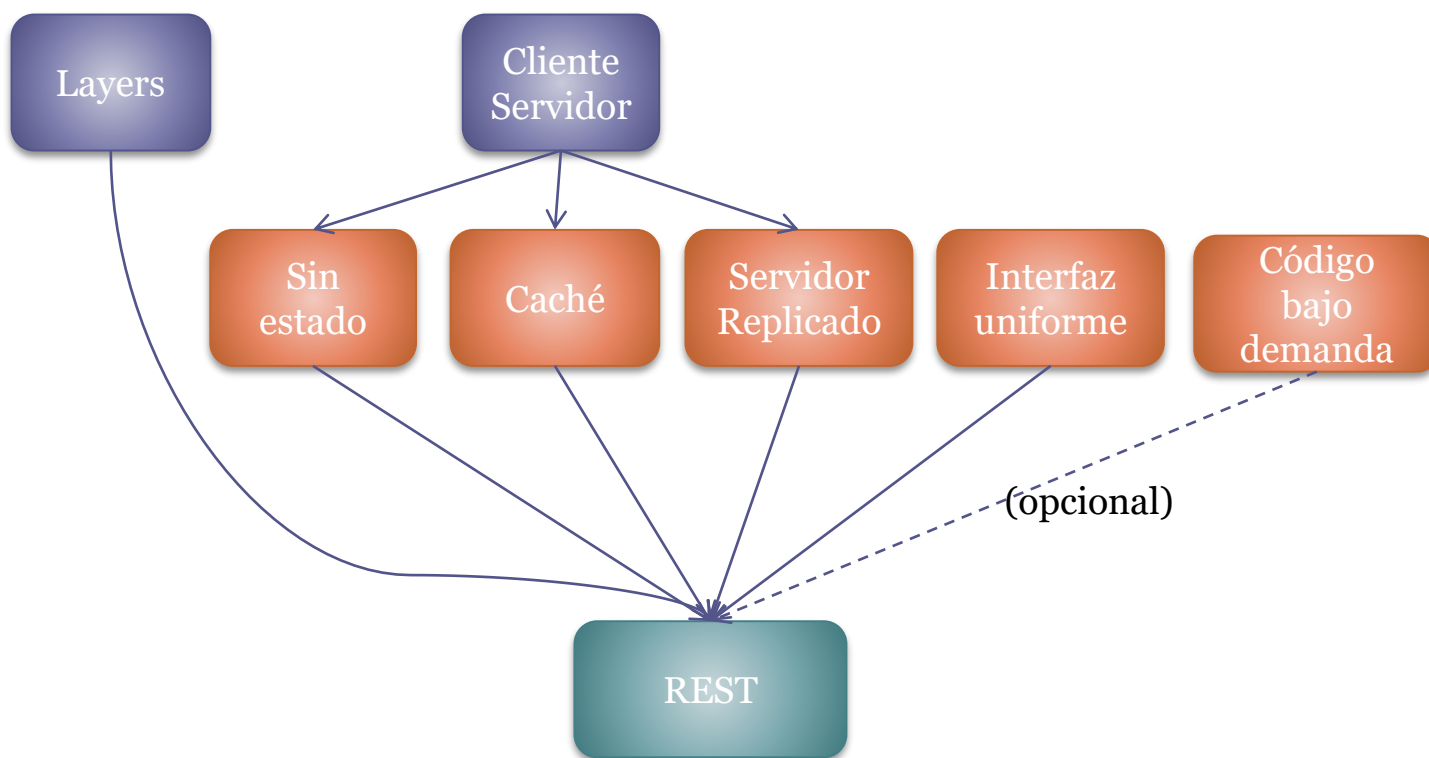
- Servicios Web sin contrato ni descripción

- REST con estilo RPC

Dificultades para incorporar otros requisitos

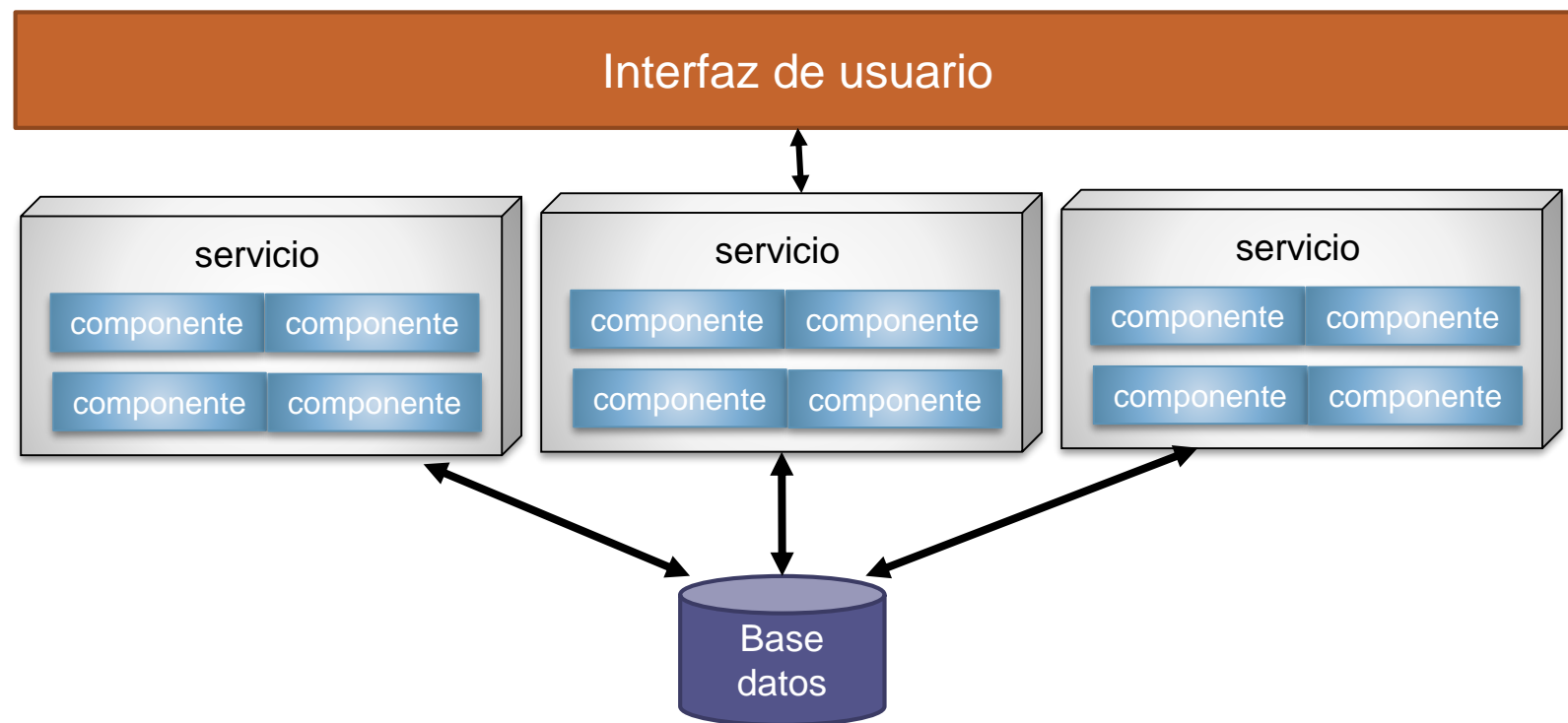
- Seguridad, transacciones, composición, etc.

REST como estilo compuesto



Arquitectura basada en servicios

Estilo arquitectónico pragmático basado en SOA



Arquitectura basada en servicios

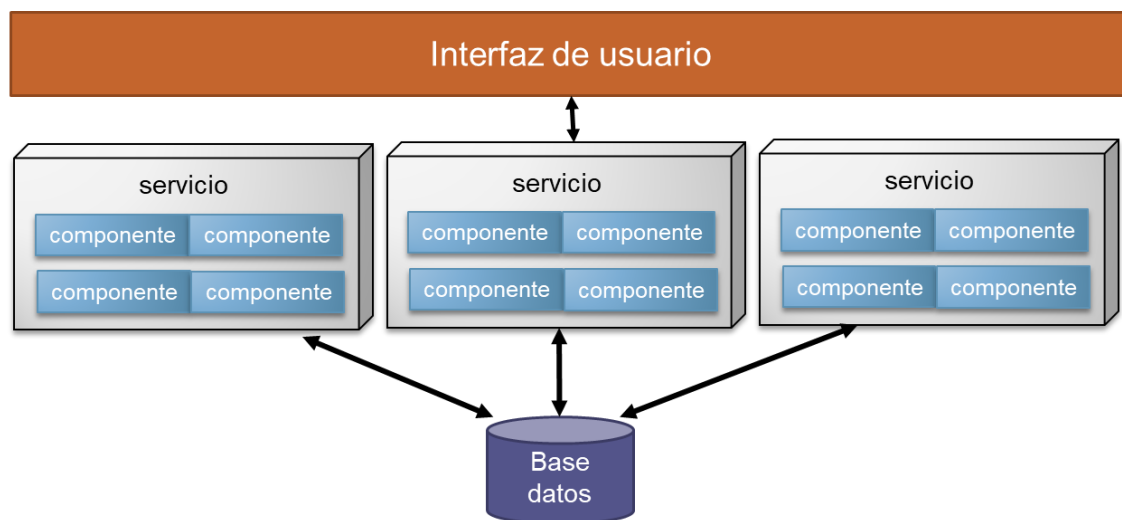
Elementos

Servicios = Unidades desplegadas
independientemente

Normalmente formados por varios componentes

El interfaz de usuario accede a servicios de forma remota (Internet)

Base de datos



Service based architecture

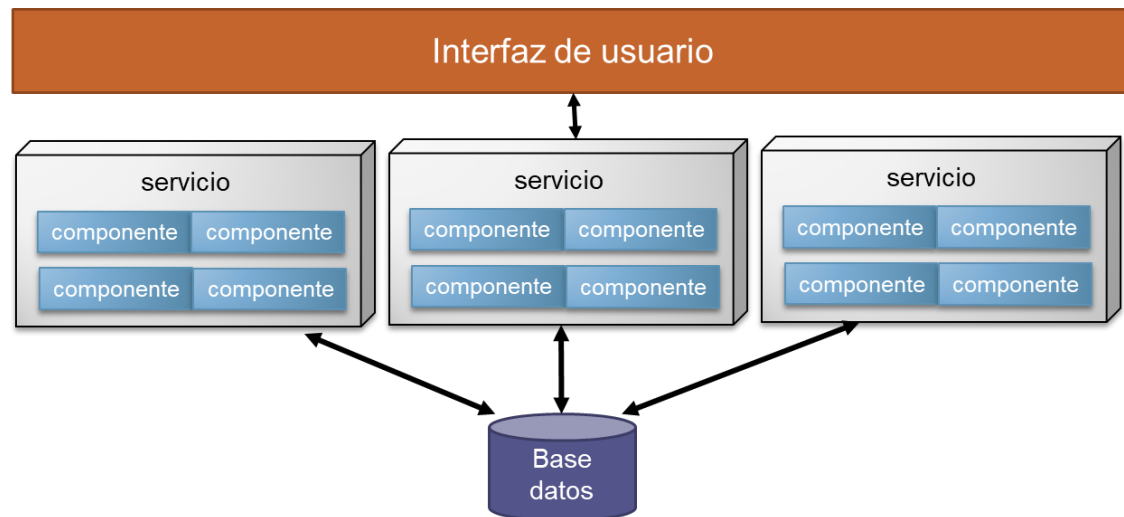
Restricciones

Cada servicio es desplegado de forma independiente

Servicios pueden ser grandes

Interfaz de usuario puede dividirse (varias topologías)

Base de datos compartida por cada servicio



Arquitectura basada en servicios

Ventajas

Modularidad de desarrollo

Servicios independientes

Diversidad tecnológica

Cada servicio puede implementarse con tecnologías y lenguajes diferentes

Time to market

Varios frameworks disponibles

Disponibilidad

Fiabilidad

Retos

Escalabilidad (particionado base datos)

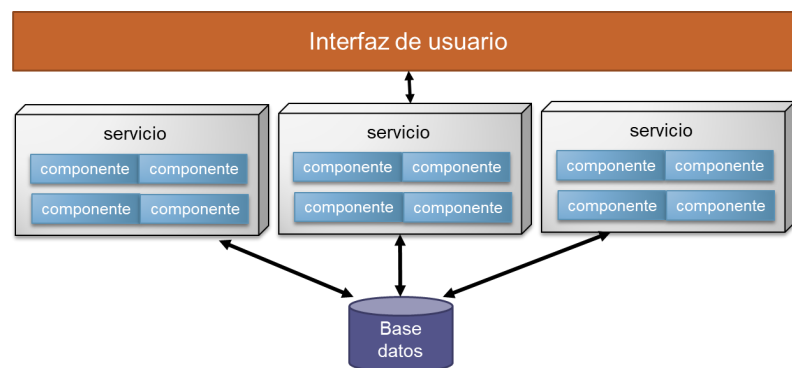
Evolución de servicios

Difícil adaptación al cambio

Servicios como monolitos

Ley de Conway

Equipos de base de datos, interfaz de usuario, desarrolladores...



Microservicios

Aplicaciones complejas se dividen en componentes, llamados servicios

Cada servicio = bloque de construcción pequeño

Altamente desacoplados

Enfocados a hacer una pequeña tarea

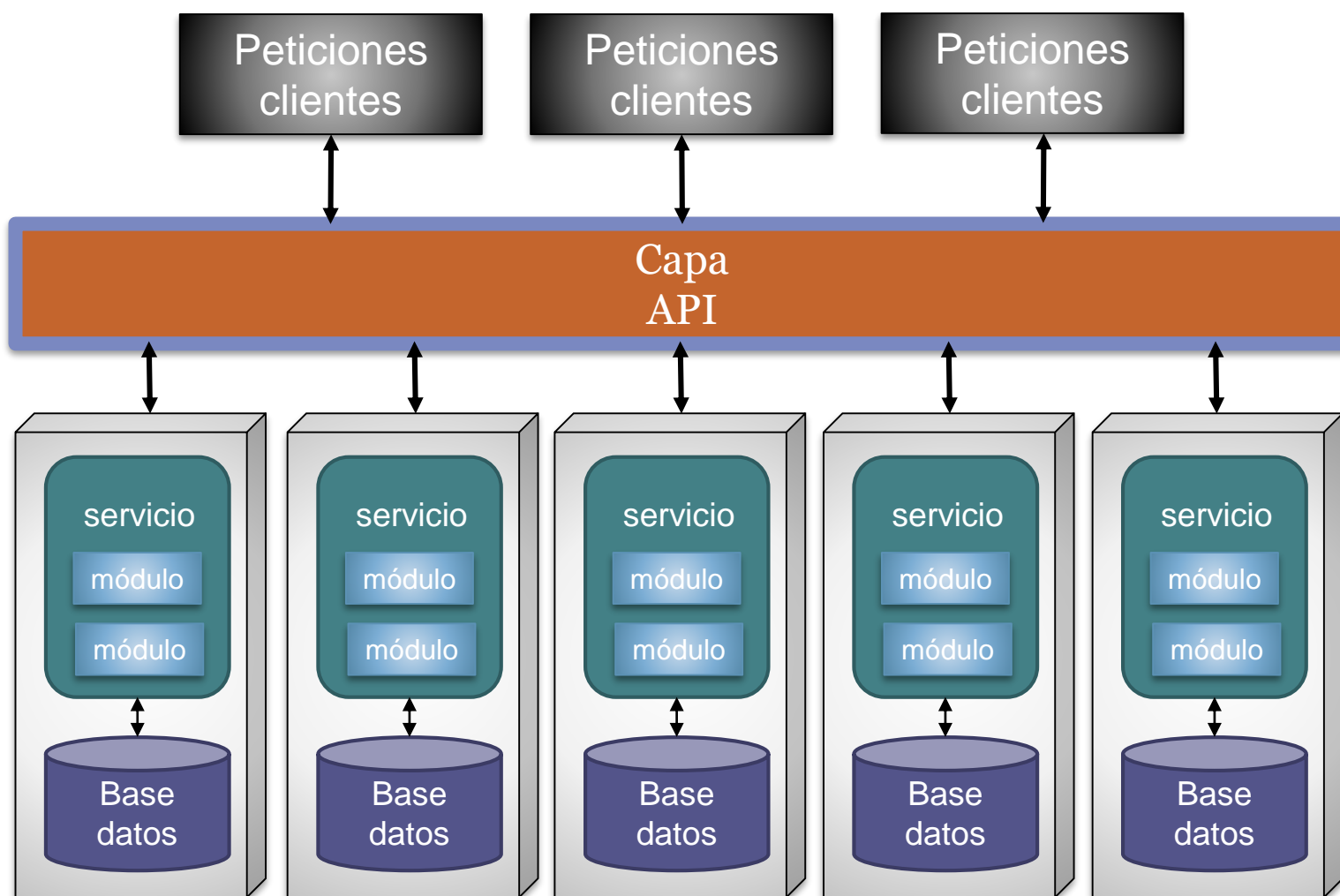
Diferencia respecto a SOA

En SOA servicios de diferentes aplicaciones

Microservicios pertenecen a una misma aplicación

Microservicios

Diagrama



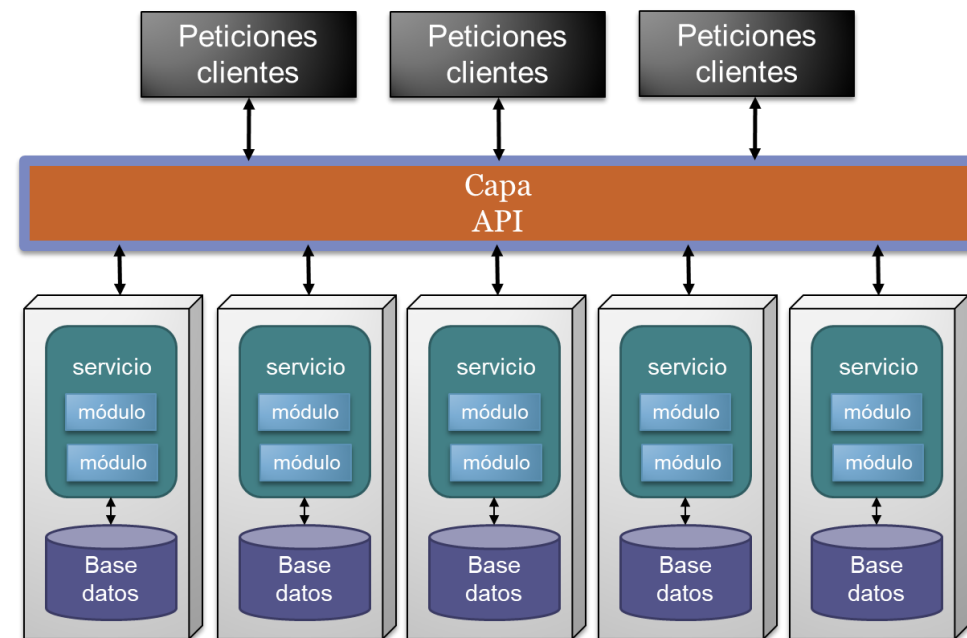
Microservices

Elementos

Component desplegado = Servicio + base datos

Servicio puede contener varios módulos

Capa API (opcional) es un proxy o servicio de nombrado



Microservicios

Restricciones

Servicios distribuidos

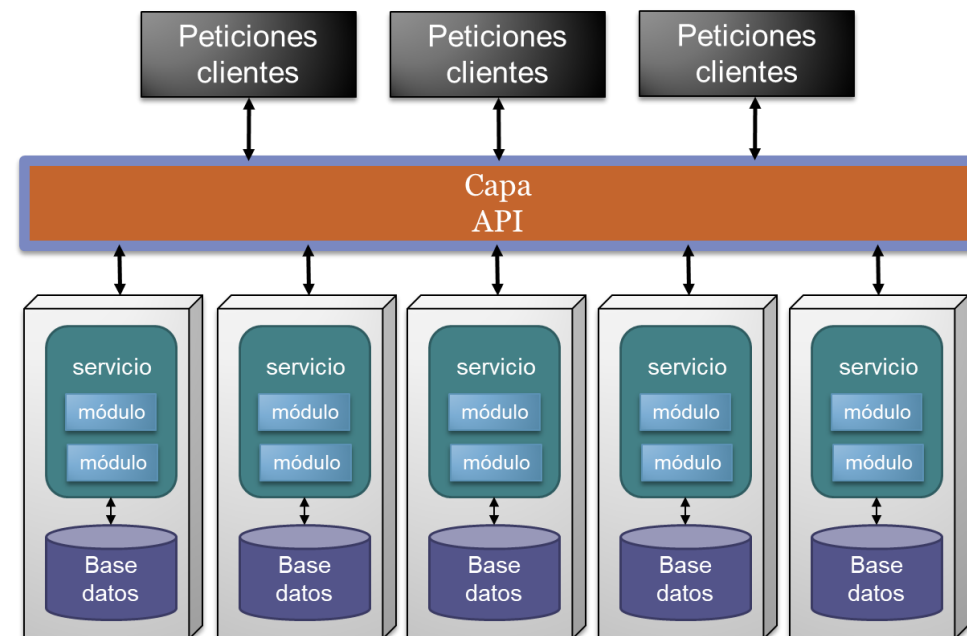
Contexto definido (bounded context):

Cada servicio modela un dominio o flujo de trabajo

Aislamiento de datos

Independencia:

No hay mediador u orquestador

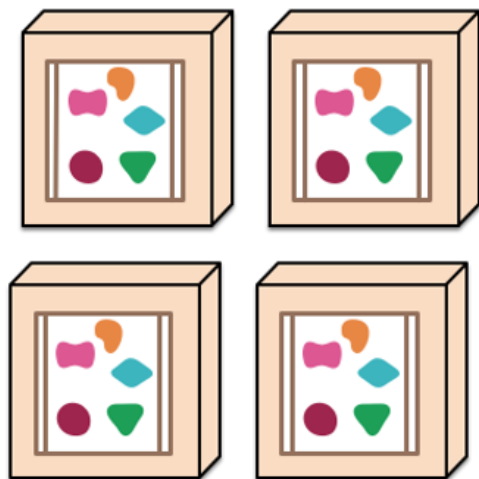


Microservicios y escalabilidad

Aplicación monolítica
Toda funcionalidad en un solo
proceso



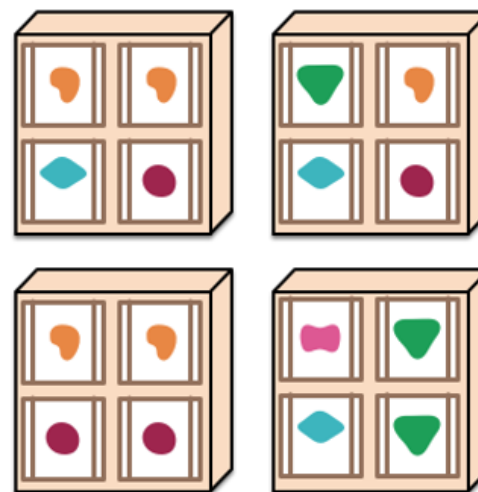
...escalabilidad mediante
replicación del monolito
en diferentes servidores



Microservicios: Cada
funcionalidad distribuida
en un microservicio

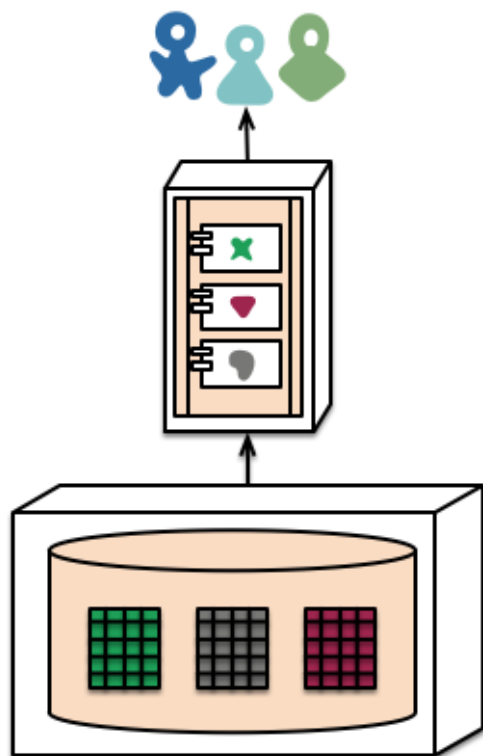


...escalabilidad mediante
distribución de servicios en
servidores y replicación

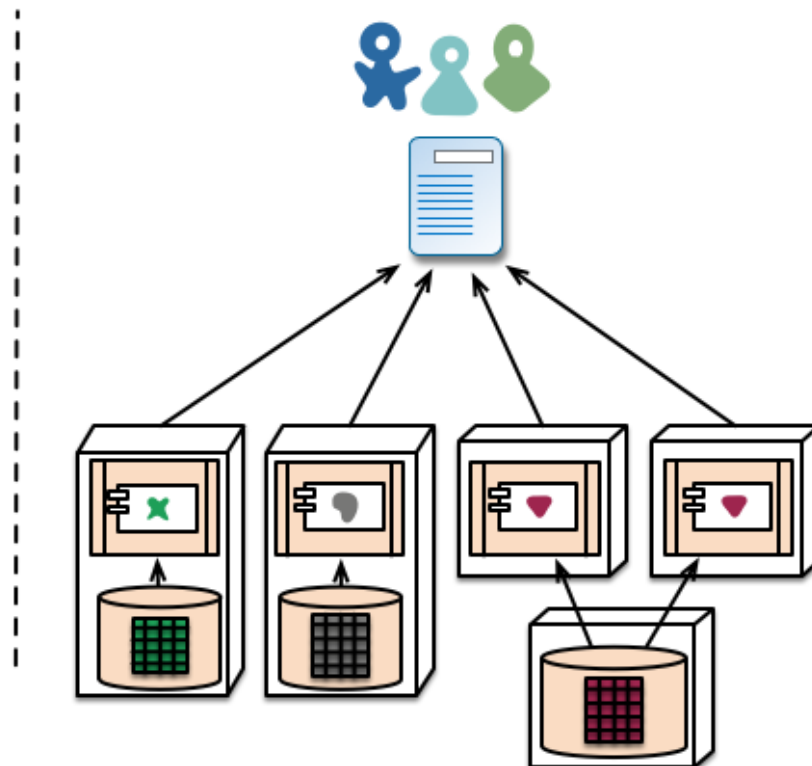


Microservicios

Gestión de datos descentralizada



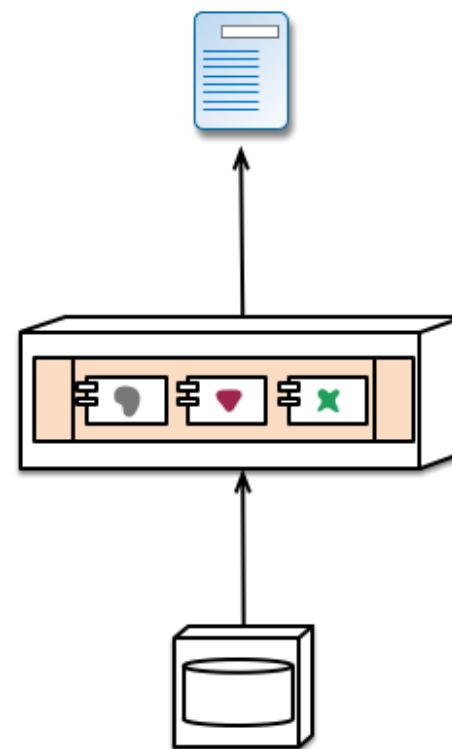
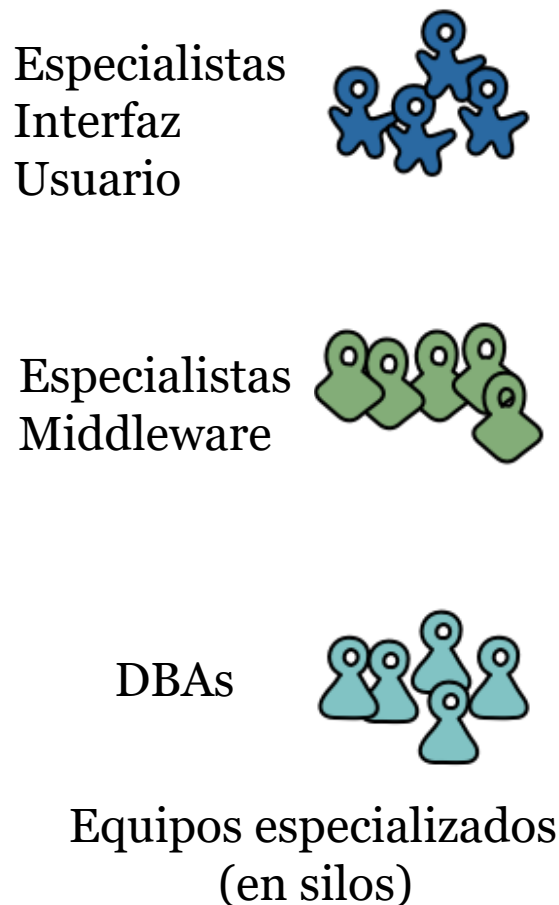
monolith - single database



microservices - application databases

Microservicios

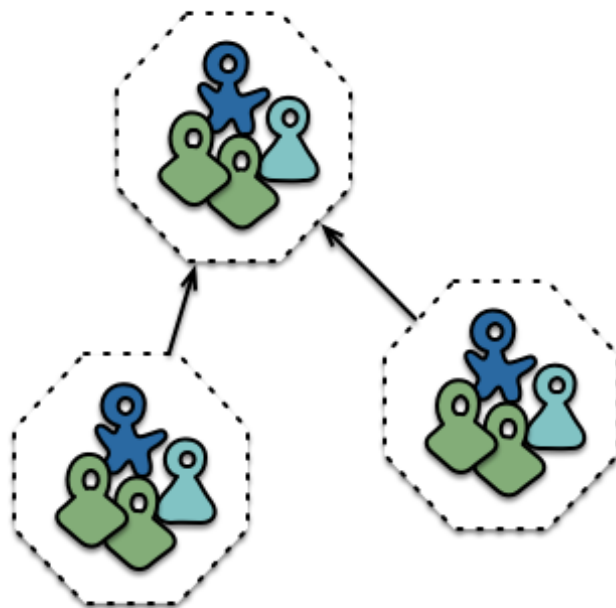
Ley de Conway en aplicación tradicional



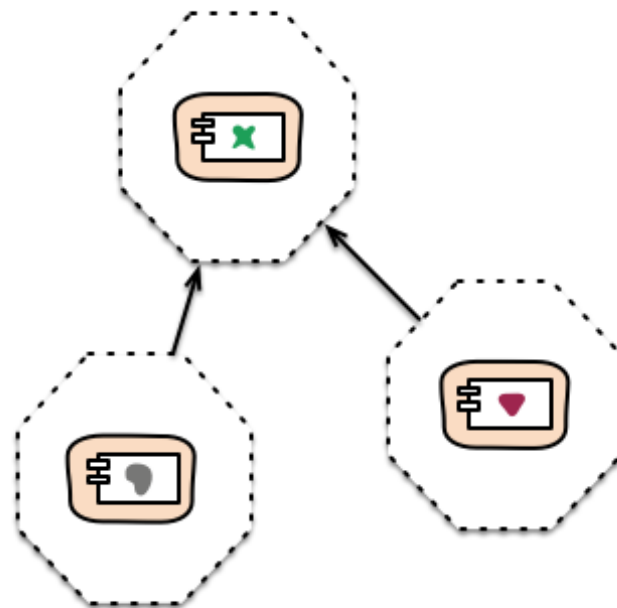
...lleva a arquitecturas basadas en silos
Debido a la Ley de Conway

Microservicios

Ley de Conway con microservicios: Equipos basados en funcionalidad



Equipos multidisciplinares
Funcionalidad cruzada



Organizados alrededor de las capacidades
Debido a la Ley de Conway

Microservicios

Ventajas

Modularización del desarrollo

Reusabilidad del microservicio

Desarrollo y despliegue independiente

Escalabilidad

Descentralización

Independencia de tecnologías concretas

Cada servicio puede desarrollarse con un lenguaje y una tecnología diferentes

Diversidad tecnológica

Problemas/retos

Gestión de muchos microservicios

Demasiados microservicios = antipatrón (nanoservicios)

Garantizar la consistencia de la aplicación

Complejidad de desarrollo

Sistemas distribuidos son difíciles de gestionar

Aparecen nuevos problemas: latencia, formato de mensajes, balance de carga, tolerancia a fallos, etc.

Pruebas y despliegue

Complejidad operacional

Deterioro estructural

Deterioro estructural microservicios

Dependencias de código entre microservicios

Demasiadas librerías compartidas

Demasiada comunicación entre servicios

Demasiadas peticiones de orquestación

Agregación de microservicios

Acoplamiento de la base de datos

Vídeo: Analyzing architecture (microservices)

<https://www.youtube.com/watch?v=U7s7Hb6GZCU>

Microservicios

Variantes

Arquitectura de sistemas auto-contenidos

Self contained Systems (SCS) Architecture

Separación de funcionalidad en muchos sistemas independientes

<https://scs-architecture.org/>

Cada Sistema auto-contenido contiene lógica y datos

Arquitectura *Serverless*

También conocido como:

Function as a service (FaaS)

Backend as a service (BaaS)

Aplicaciones dependen de servicios de terceras partes

Los desarrolladores no tienen que preocuparse de los servidores

Escalabilidad automática

Clientes ricos

Aplicaciones Single Page, Aplicaciones móviles

Ejemplos:

AWS Lambda, Google Cloud Functions, Ms Azure Functions

https://en.wikipedia.org/wiki/Serverless_computing

Arquitectura *Serverless*

Ventajas

Escalabilidad

Disponibilidad

Rendimiento

Costes reducidos

Costes operacionales

Sólo se paga por los
recursos

computacionales
requeridos

Time to market reducido

Retos

Dependencia de un vendedor

Vendor lock-in

Incompatibilidad entre
soluciones de diferentes
vendedores

Seguridad

Latencia de arranque

Pruebas de integración

Monitorización/depuración

Fin