

ARQUITECTURA ORIENTADA A SERVICIOS - SOA

DSD



Arquitectura?

- Componentes
- Relaciones
- Mensajes
- Plan para construir
- Requisitos
- Actual y a futuro

Definiciones: Software Engineering Institute (SEI)
www.sei.cmu.edu/architecture/definitions.html.



Propósito de una Arquitectura

- Un propósito fundamental de la arquitectura de software es ayudar a gestionar la complejidad de los sistemas de software y las modificaciones que inevitablemente experimentan los sistemas en respuesta a cambios externos en los entornos empresarial, organizativo y técnico.

Estilos de Arquitectura

- Un estilo arquitectónico contiene un conjunto bien definido de patrones que constituyen una forma común para que los componentes de la solución empresarial interactúen entre sí.



Departamental a Distribuido

Tener procesos distribuidos significa que se necesita de:

- Planificación distribuida
- diseño distribuido
- realización distribuida
- operación distribuida.

La **integración y la distribución** son cada vez más importantes

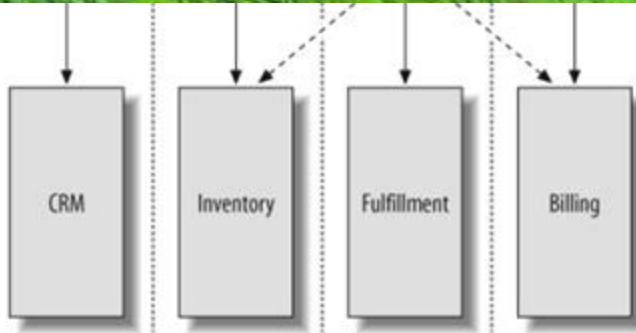


Figura 8-1. Departamentos conducen a sistemas monolíticos.

Estilos de Arquitectura

Monolítico

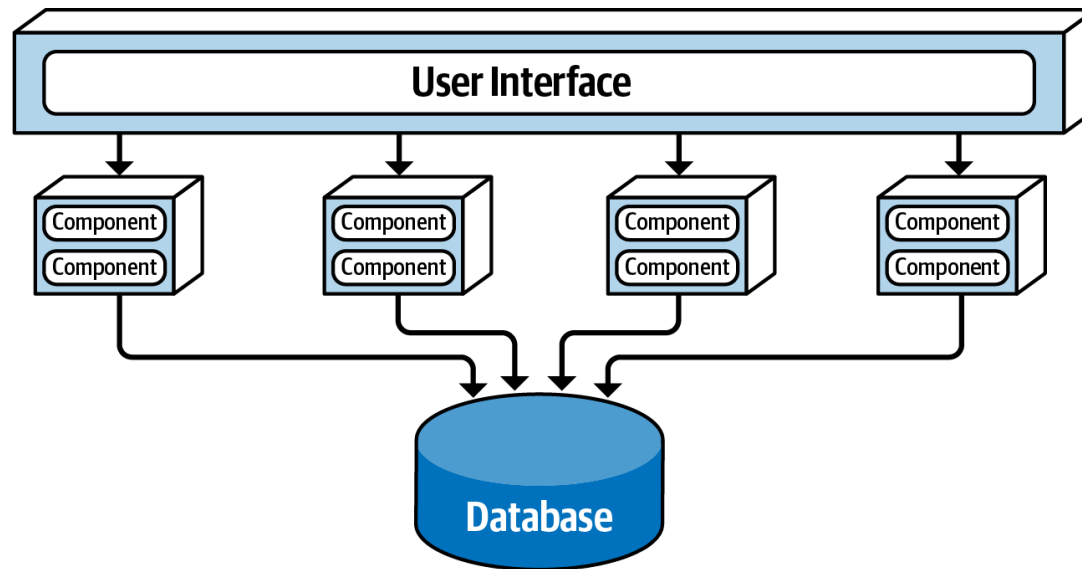
- A. Arquitectura en capas
- B. Arquitectura de canalización
- C. Arquitectura de micronúcleo

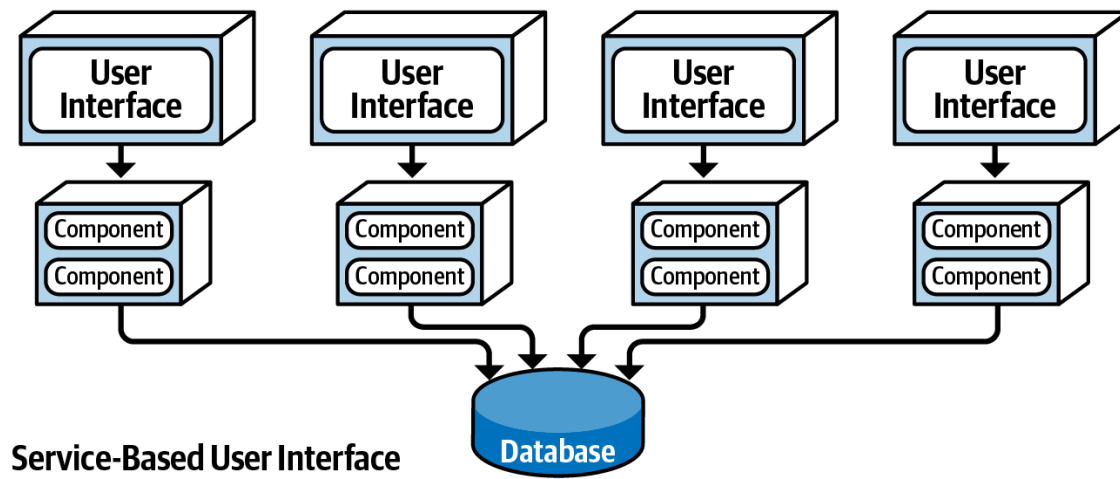
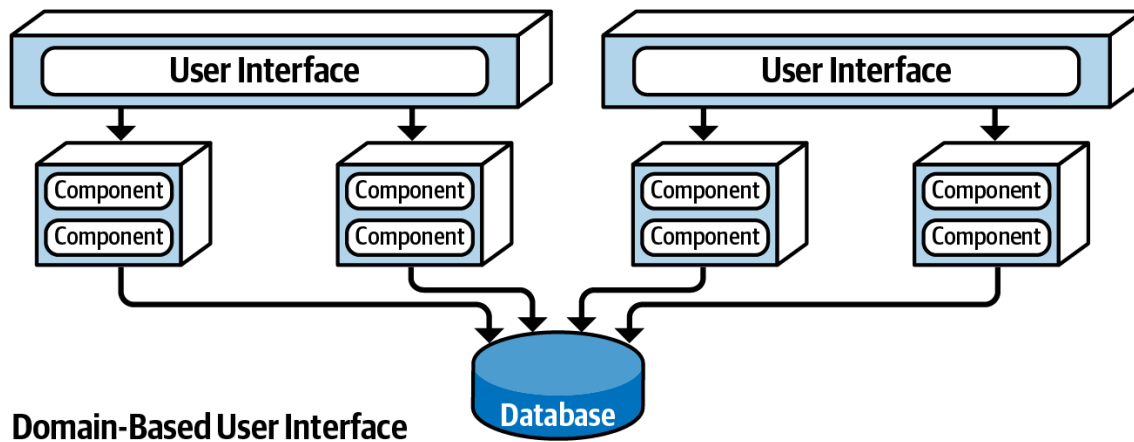
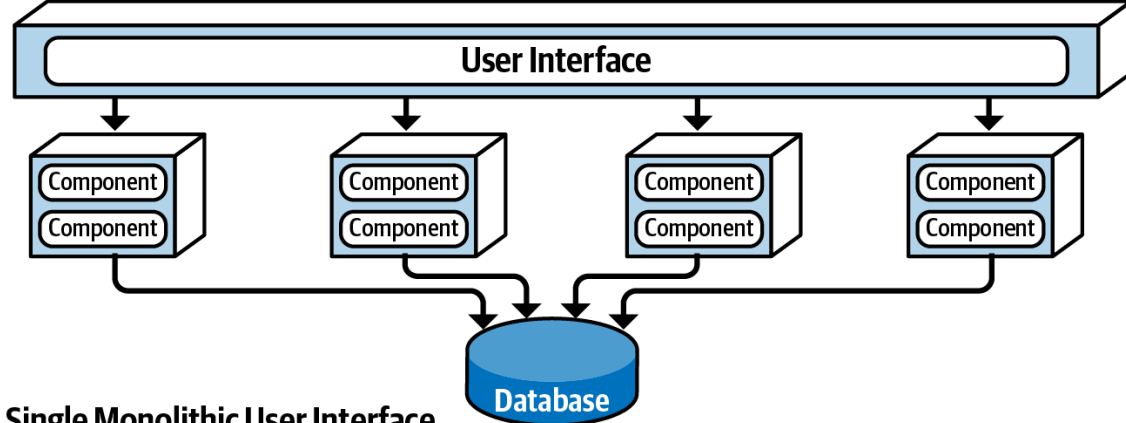
Distribuido

- 1. Arquitectura basada en servicios
- 2. Arquitectura impulsada por eventos
- 3. Arquitectura basada en el espacio
- 4. Arquitectura orientada a servicios
- 5. Arquitectura de microservicios

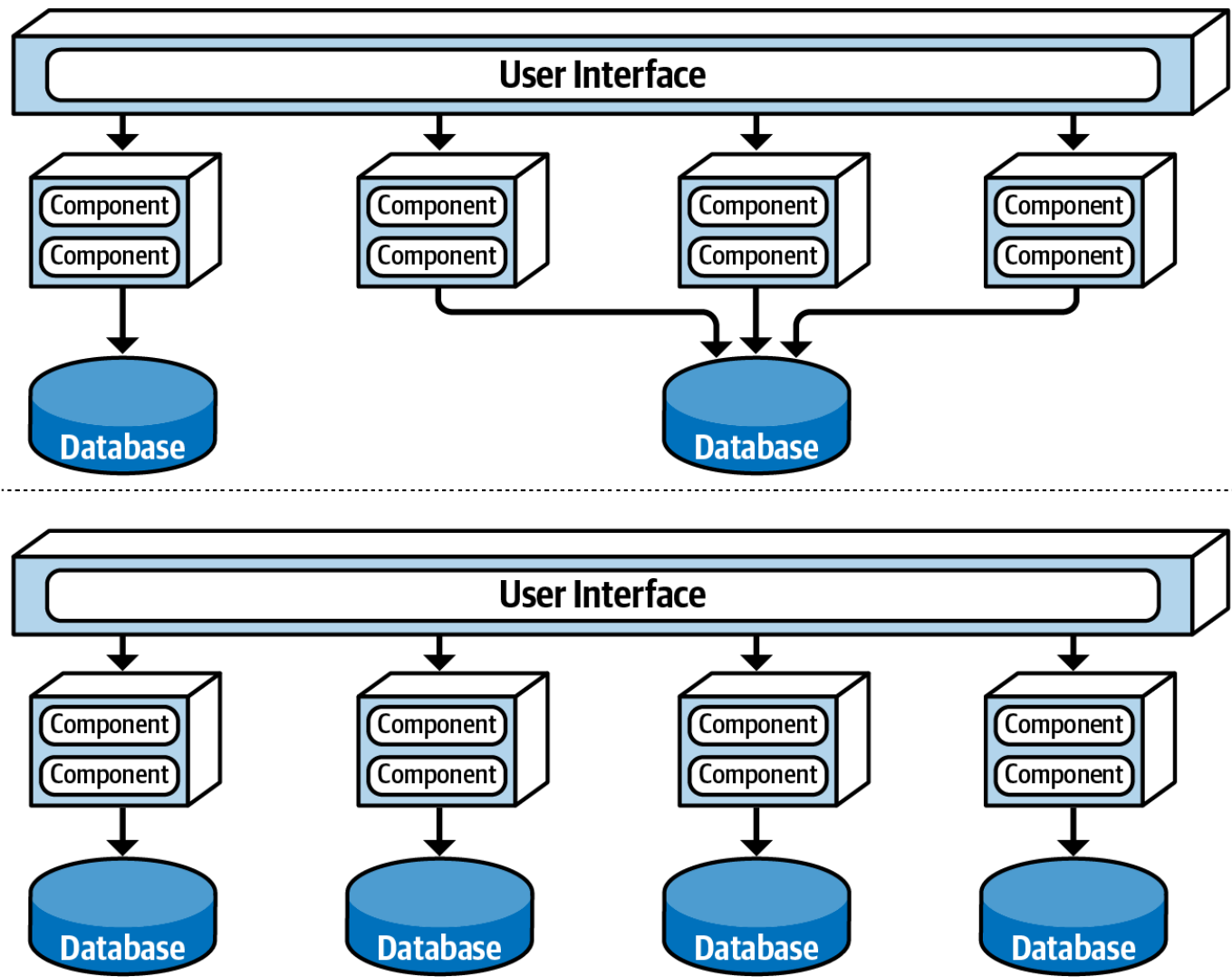
1. Arquitectura basada en servicios

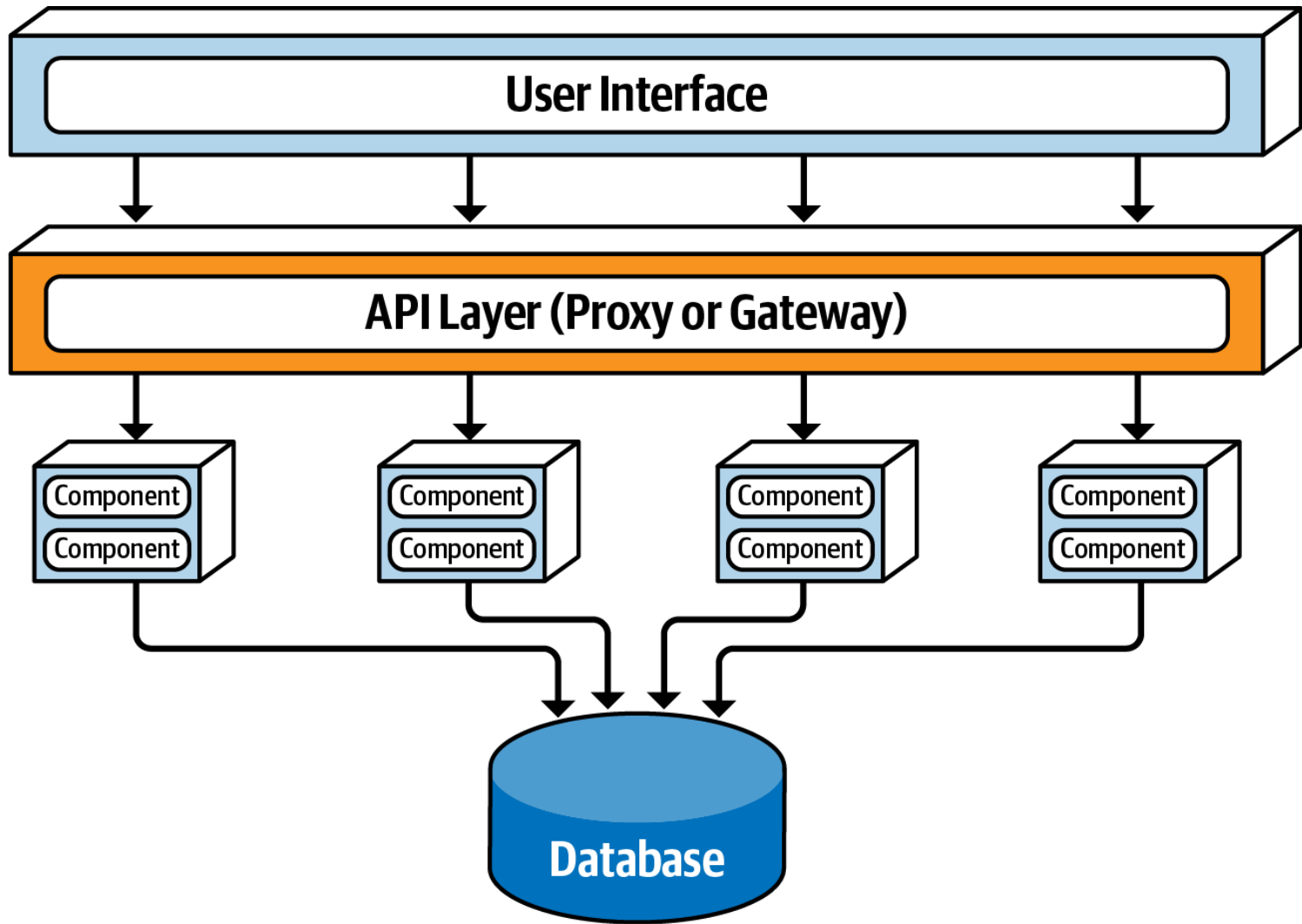
- En la mayoría de los casos, solo hay una única instancia de cada servicio de dominio dentro de una arquitectura basada en servicios.
- Sin embargo, según las necesidades de escalabilidad, tolerancia a fallas y rendimiento, ciertamente pueden existir múltiples instancias de un servicio de dominio.
- Se accede a los servicios de forma remota desde una interfaz de usuario utilizando un protocolo de acceso remoto.



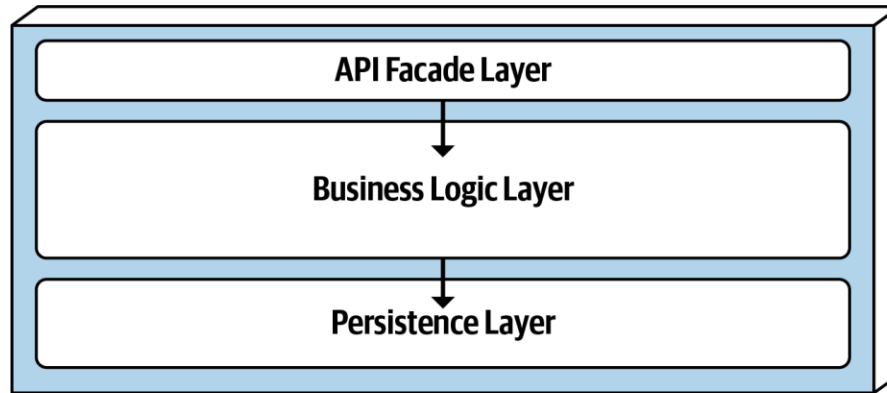


Base de datos separadas:



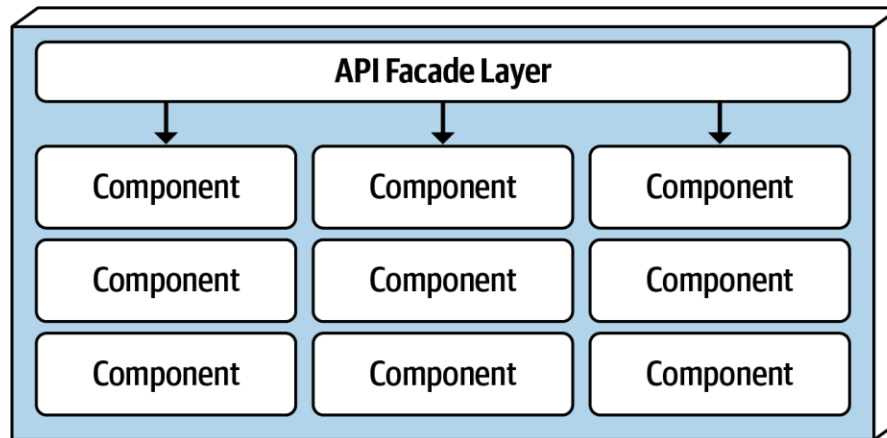


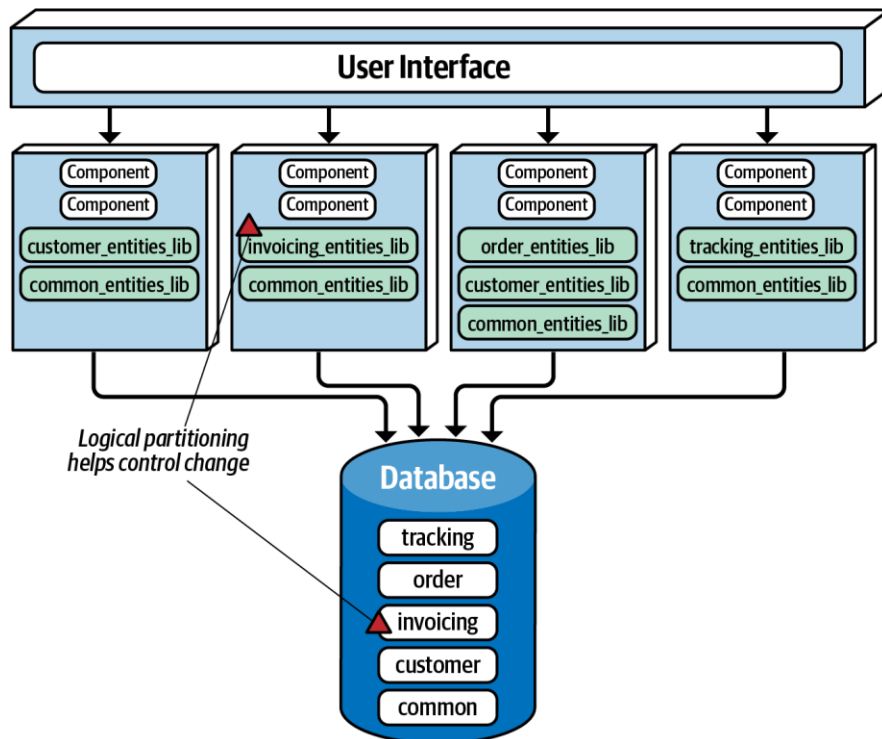
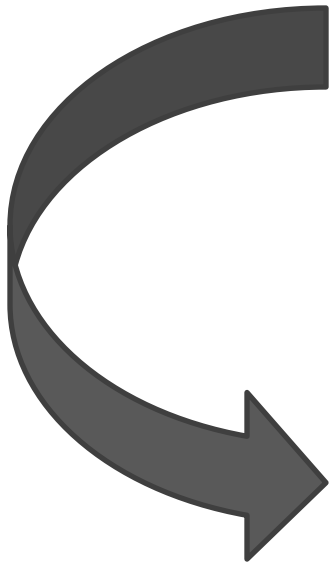
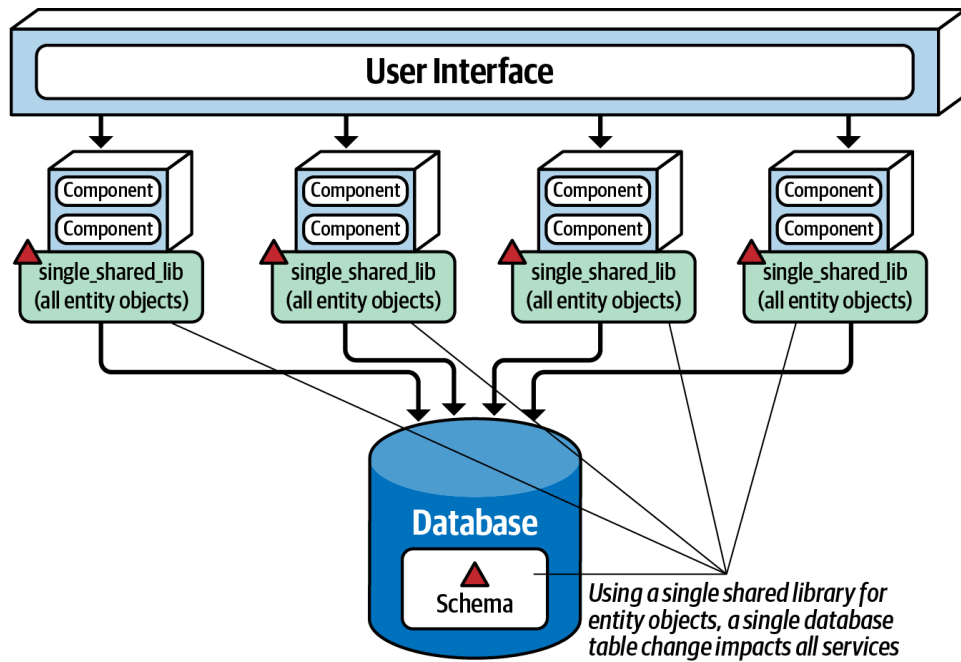
Diseño de Servicio - Granularidad



Layered design (technical partitioning)

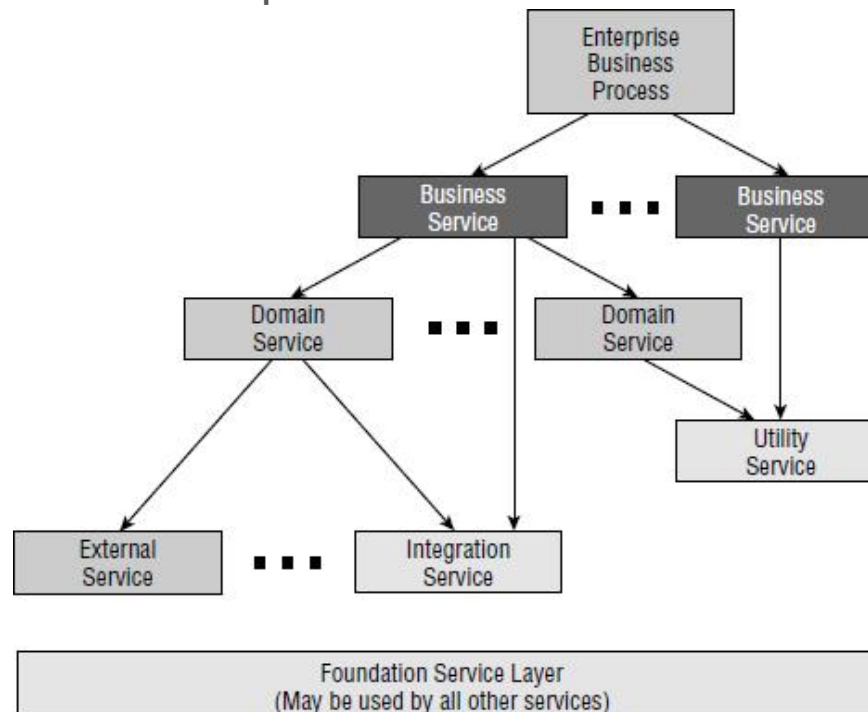
Domain design (domain partitioning)





Granularidad del Servicio

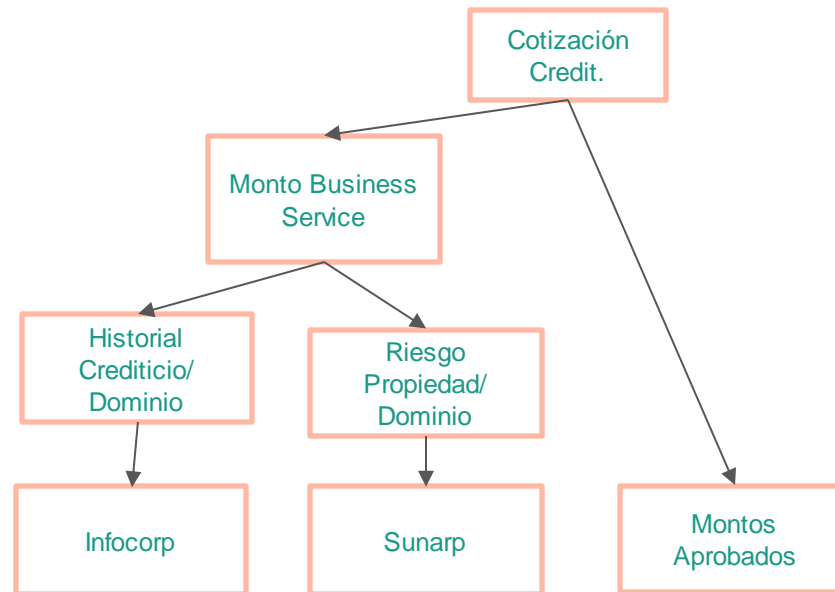
- Cantidad de función comercial que se realiza en un solo intercambio de mensajes de solicitud / respuesta.



Ejemplo

Crear un nuevo proceso comercial para para aprobar cotizaciones ante solicitudes de crédito

- Determinar su deuda o historial crediticio en Infocorp
- Determinar el riesgo asociado a sus propiedades
- Determinar el tipo de crédito ofrecido asociado a los solicitado

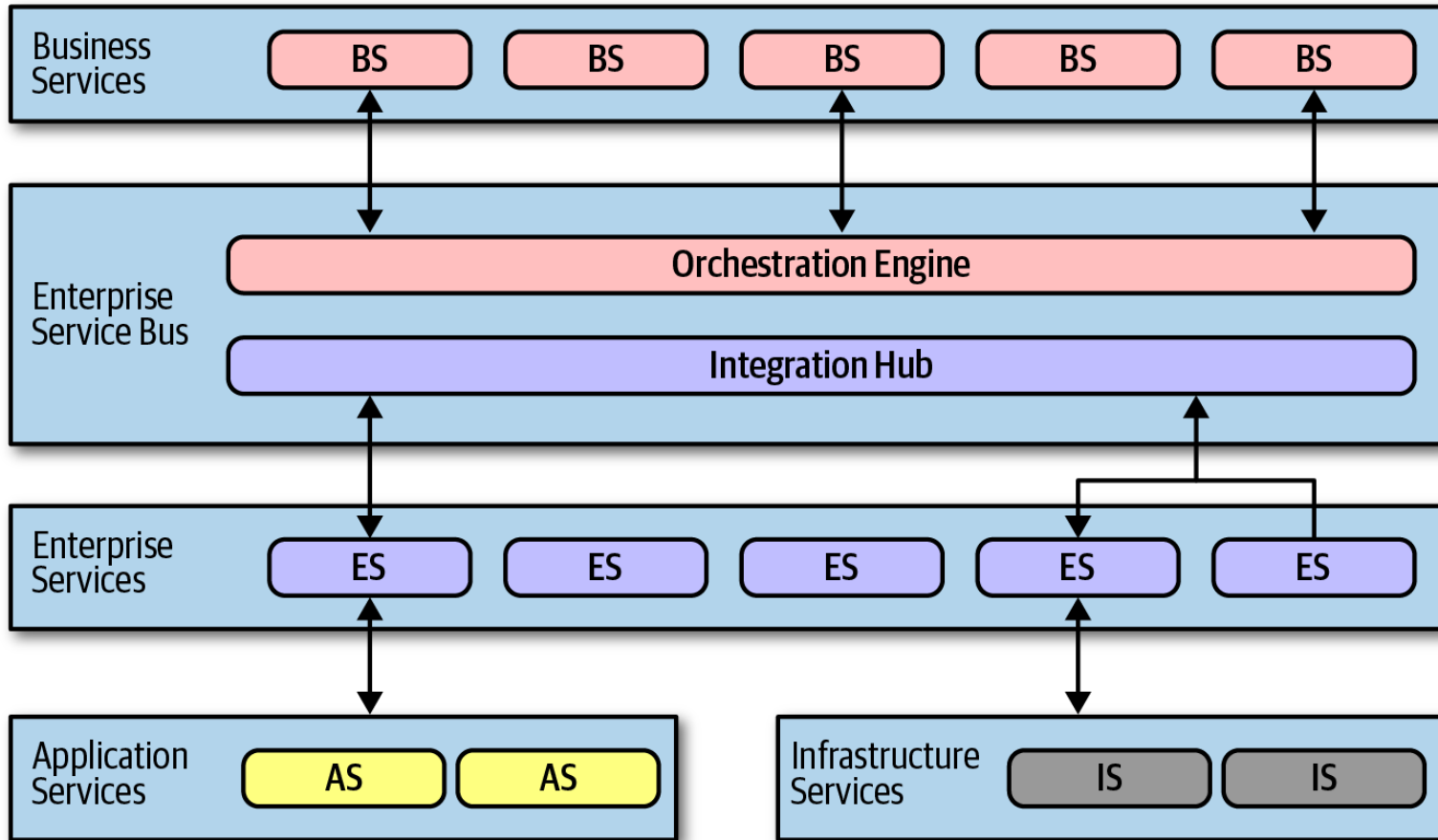




Ventajas de la Arquitectura Basada en Servicio

- Es una buena opción cuando se realiza un diseño basado en dominios
- Preserva las transacciones ACID mejor que cualquier otra arquitectura distribuida debido a la naturaleza de grano grueso de los servicios de dominio.
- Hay casos en los que la interfaz de usuario o la puerta de enlace API pueden orquestar dos o más servicios de dominio y, en estos casos, la transacción debería depender de sagas y transacciones BASE. Sin embargo, en la mayoría de los casos, la transacción se limita a un servicio de dominio en particular.
- Por último, la arquitectura basada en servicios es una buena opción para lograr un buen nivel de modularidad arquitectónica sin tener que enredarse en las complejidades y trampas de la granularidad.

3. Arquitectura orientada a servicios impulsada por orquestación

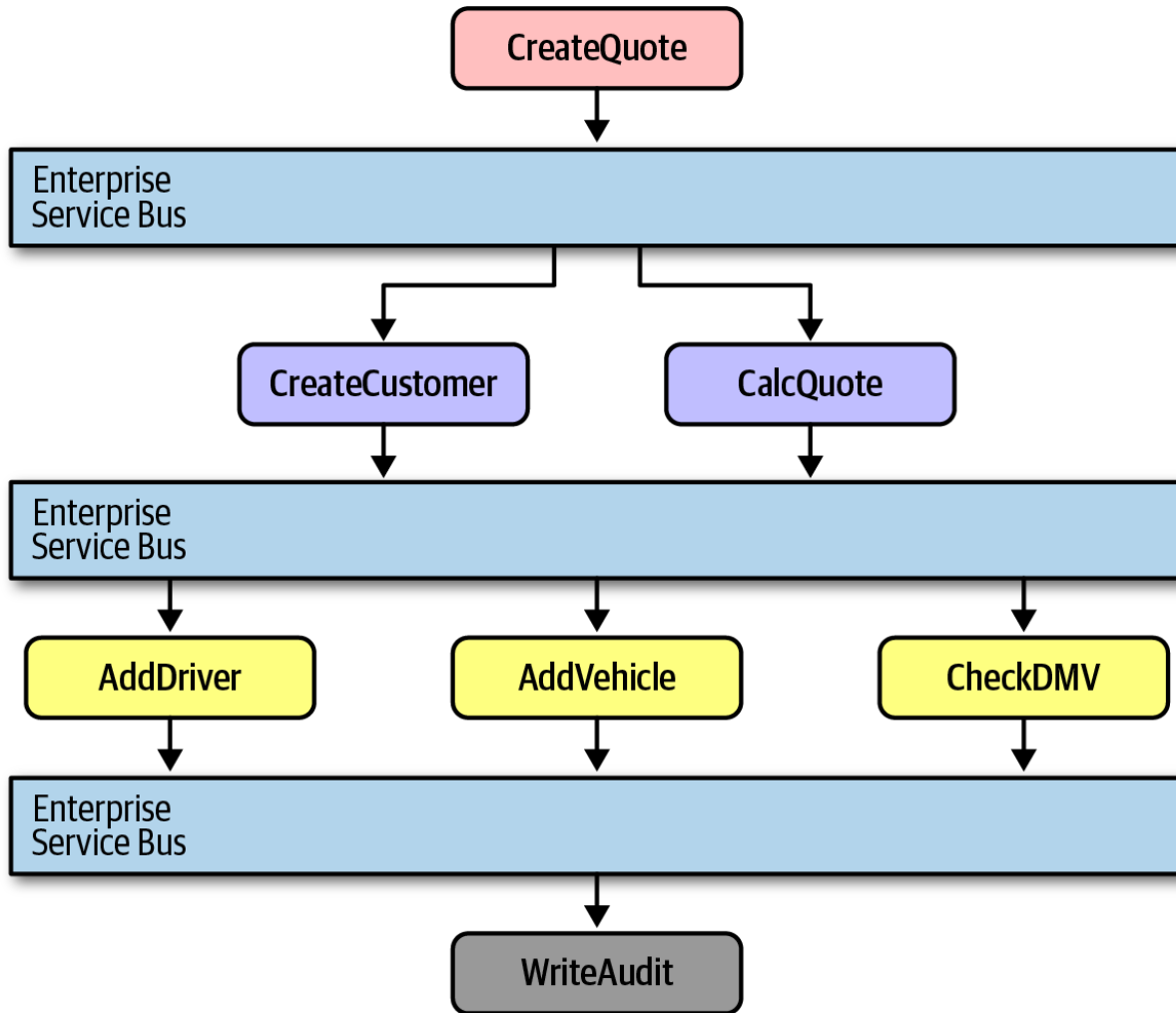


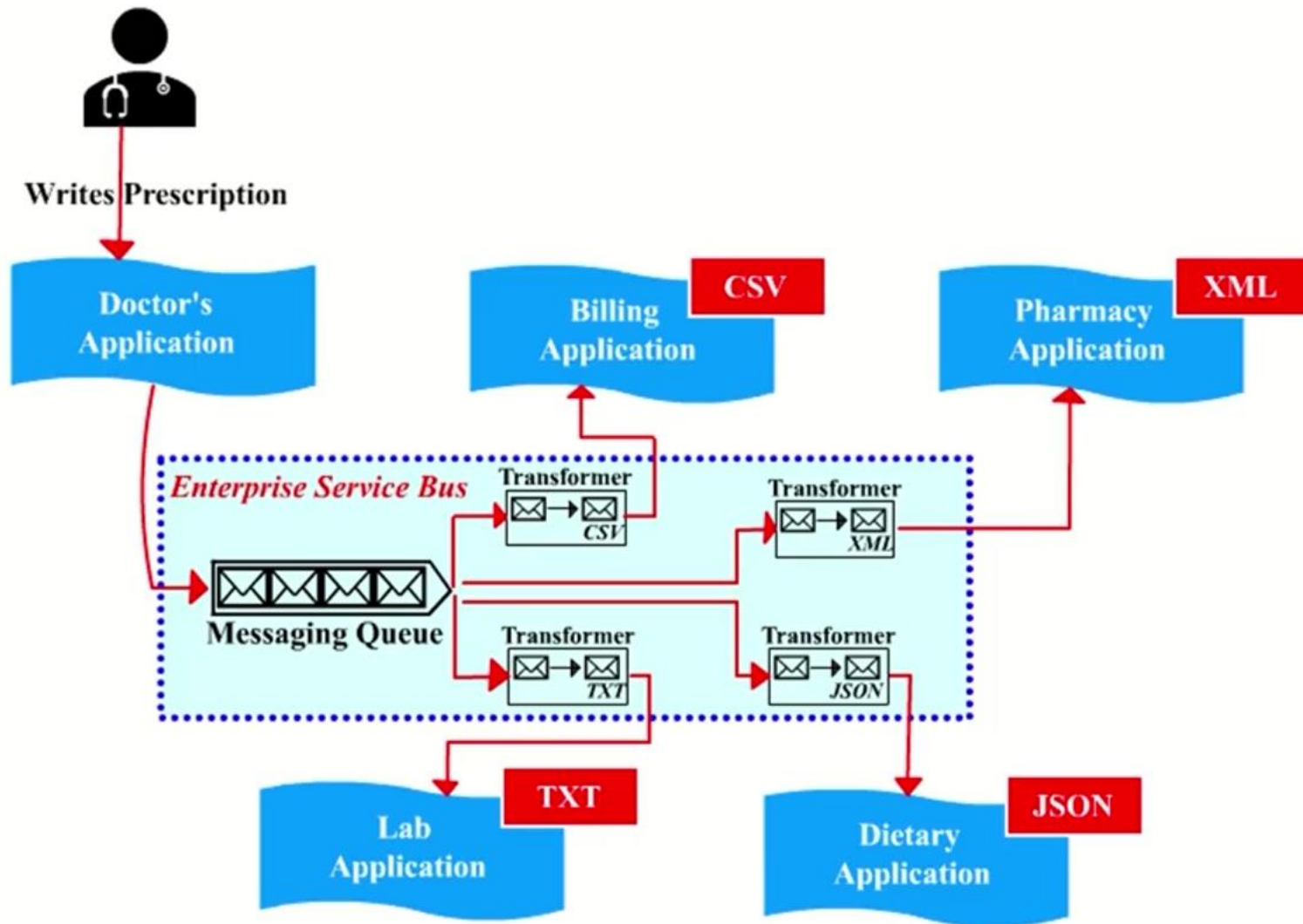
SOA (Service Oriented Architecture)

“Estilo arquitectónico que promueve el concepto de un **servicio** empresarial **alineado con el negocio** como la unidad fundamental de diseño, construcción y composición de soluciones empresariales.
Múltiples patrones que describen definiciones, implementaciones y despliegue de las soluciones SOA completan este estilo.”

“SOA ofrece un marco de trabajo para **alinear los procesos de negocio con los sistemas** de TI.”

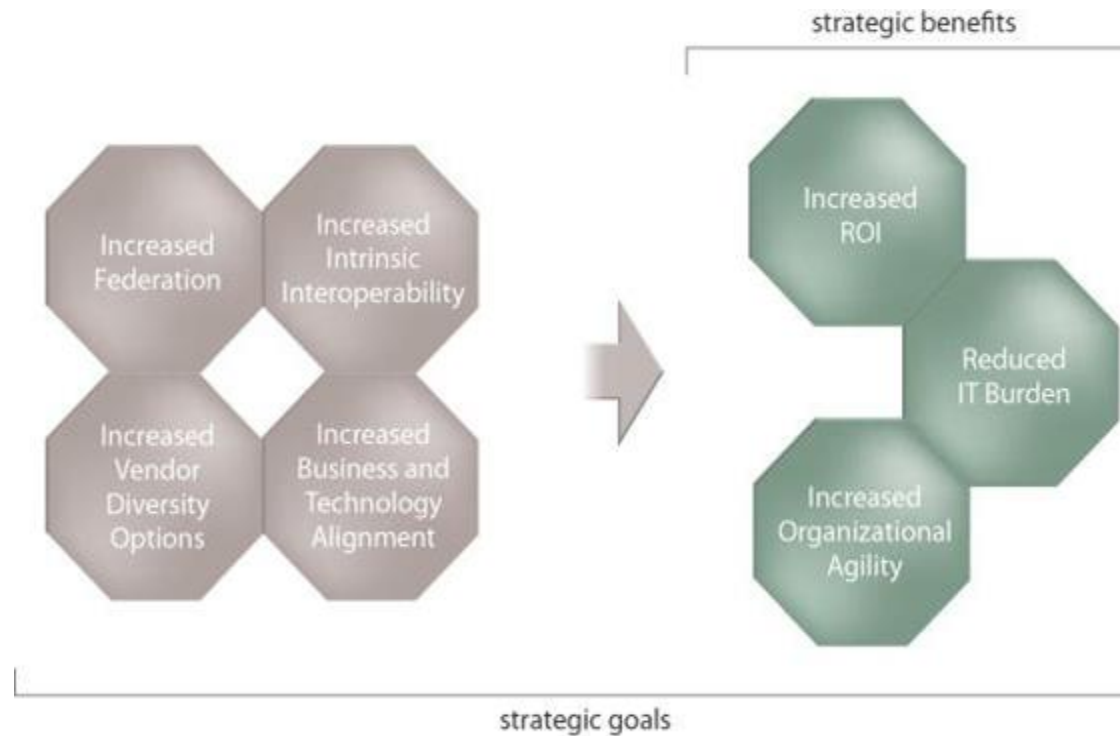
Flujo de Mensajes





Ventajas	Desventajas
Procesadores de eventos altamente desacoplados	Control de flujo de trabajo
Alta escalabilidad	Manejo de errores
Alta capacidad de respuesta	Recuperabilidad
Alto rendimiento	Capacidades de reinicio
Alta tolerancia a fallas	Inconsistencia de datos

OBJETIVOS ORGANIZACIONALES

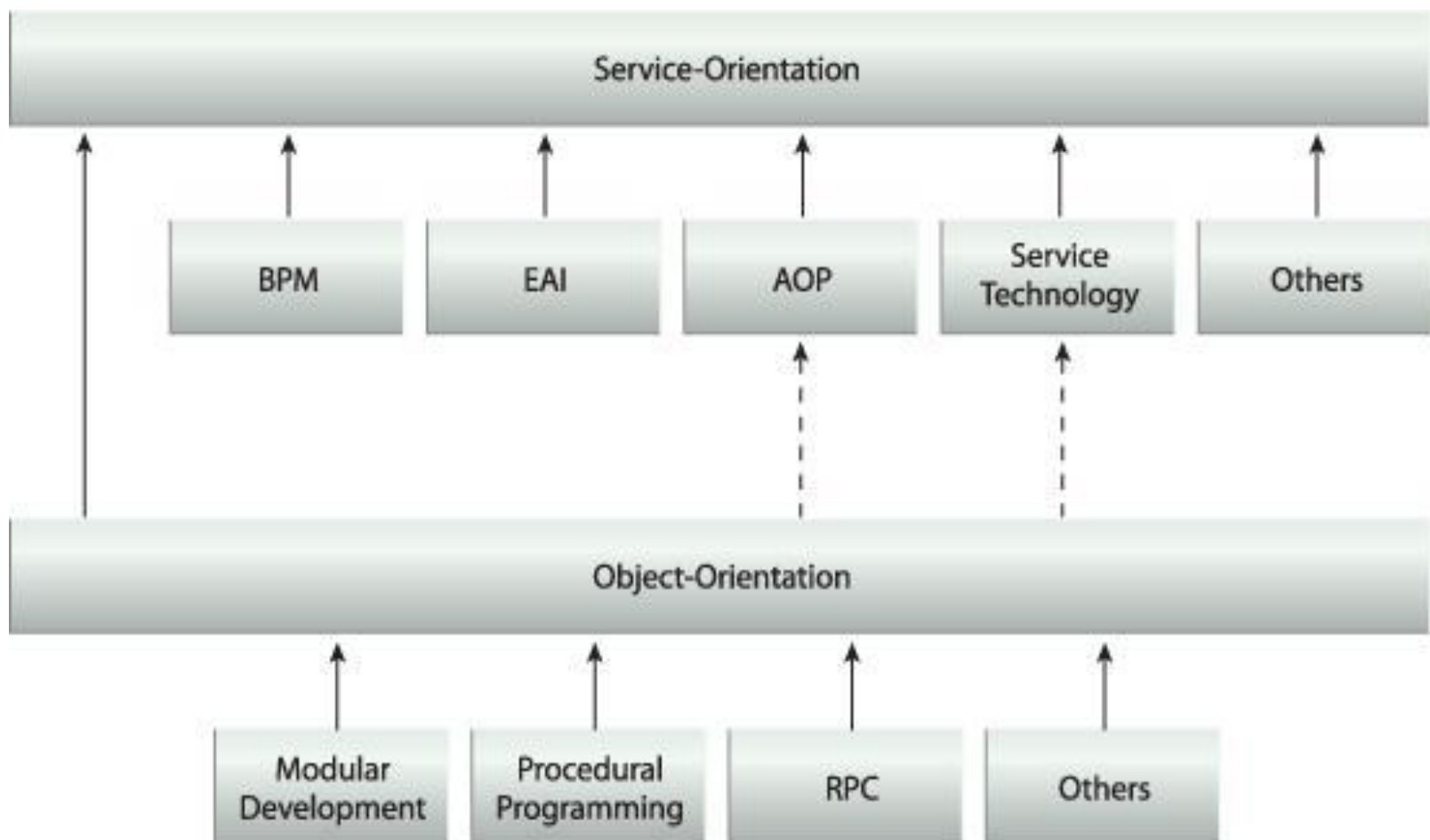


- Mayor interoperabilidad intrínseca
- Mayor federación
- Mayores opciones de diversificación de proveedores
- Mayor alineación empresarial y tecnológica

- Mayor ROI
- Mayor agilidad organizativa
- Reducción de la carga de TI

Reutilización... y acoplamiento

- Un objetivo principal de esta arquitectura es la reutilización a nivel de servicio: la capacidad de construir gradualmente el comportamiento empresarial que se puede reutilizar de forma incremental con el tiempo.



Concepciones erróneas sobre SOA

- Basta utilizar Web Services para estar "service-oriented"
- SOA es una moda o estrategia comercial para renombrar los Web Services
- SOA es una simplificación de la computación distribuida
- Basta entender Web Services para implementar SOA
- Basta SOA para lograr que todo sea interoperable



SOA Diseño

Como encontramos servicios?

Top Down - ADI

1. Stake Holders
2. Business Functions
Organigrama (Est. Fija)
1. Business Process (Est. Dinámica)
2. Business Entities
3. Plan Estratégico – Visión/Misión

1. Transacciones
2. Aplicaciones
3. Fuente de Datos

Bottom Up



Visión consumidor

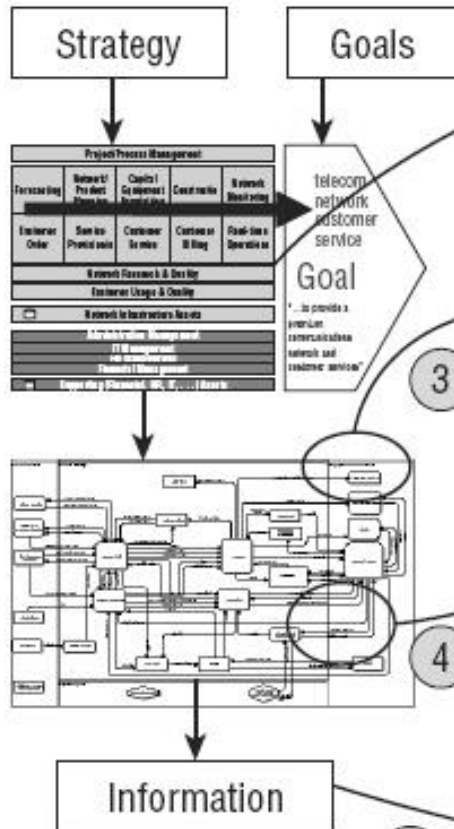
Basado en la Necesidad



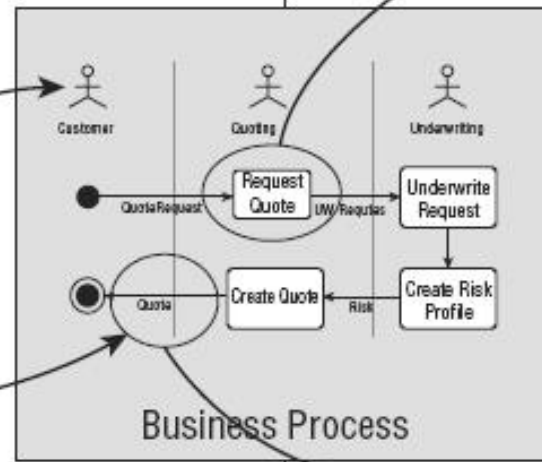
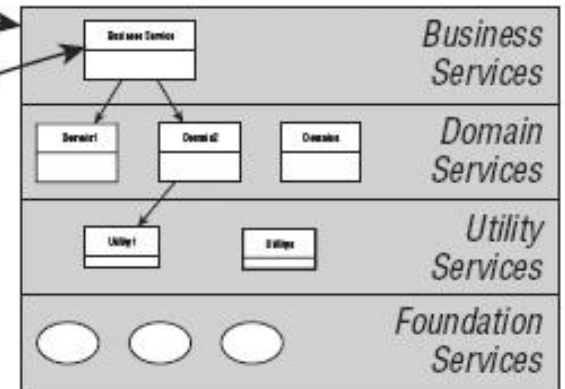
Visión Proveedor

Técnicos, expertos, juicio expertos, conocen las transacciones, juntarlos y digan que ofrecer, no hay mucho A/D

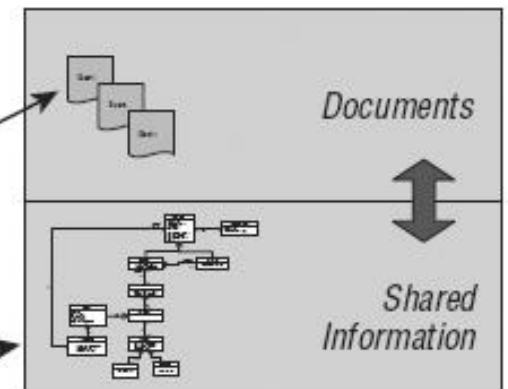
Business Architecture



SOA Business Model



Service Model/Inventory



Information Model



ORIENTACION A SERVICIOS SOA

Agenda

- ❖ Orientación a Servicios
- ❖ Principios de Orientación a Servicio
- ❖ SOA
- ❖ Arquitectura
- ❖ Tipos de Arquitectura
- ❖ Concepciones erróneas sobre SOA
- ❖ SOA Diseño
- ❖ Descomposición de Procesos en Servicios
- ❖ Ejercicios

Orientación al Servicio

La orientación al servicio es una forma de pensar en términos de servicios y desarrollo basado en servicios y los resultados de los servicios.



Orientación a Servicios

- Service Orientation, es un paradigma de diseño destinado a la creación de unidades lógicas de solución que tienen forma individual para que puedan usarse de forma colectiva y repetida en apoyo de las realizaciones de los objetivos estratégicos de la empresa.

- Servicios

- Las unidades de lógicas de la solución se denominan “Servicios”
 - El paradigma de diseño de Orientación a Servicios se compone de ocho principios planteados por Thomas Erl.

Servicio

- Un "Servicio" es (idealmente) una **función comercial** autónoma que acepta una o más solicitudes y devuelve una o más respuestas a través de una interfaz estándar bien definida.



Clasificación de los Servicios

✓ Servicios Basicos:

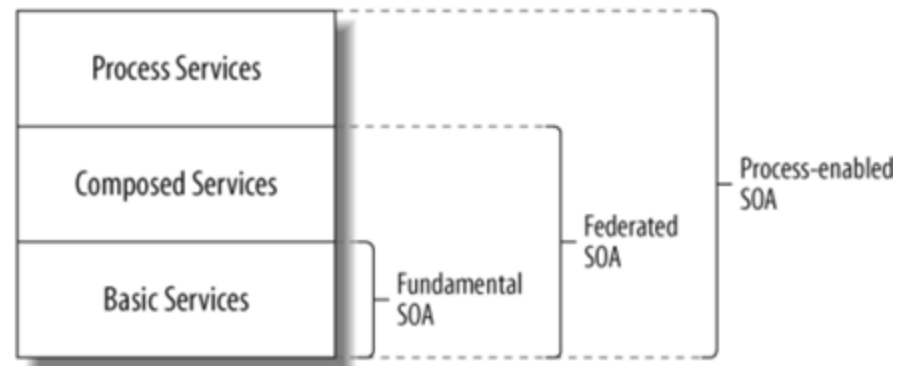
1ra etapa de expansión, SOA
Fundamental con una sola capa de servicios básicos (KrafzigBankeSlama, 2008)

✓ Servicios Compuesto:

2da etapa de expansión, SOA
Federada, capa de servicios básicos + capa de servicios compuestos (capa de orquestación o composición)

✓ Servicios de Proceso:

3era etapa de expansión SOA
habilitada para procesos + capa adicional de servicios de procesos.



Arquitectura SOA- Detalle Clasificación de los Servicios

1) BASIC SERVICE LAYER

- Capa Agnóstica, Genérica

- Capa Entity: Capa que controla su propia entidad en la base de datos, generalmente los CRUDs o mantenimientos de su tabla, interactúa directamente con su propia base de datos local.
- Capa Integration: Capa que invoca servicios externos, tales como webservices, servicios REST, servicios SAP, o legacy systems, también se consideran a los envíos de mensajes a servidores de correos, sms, servicios asíncronos de colas entre otros. Esta capa no tiene logica de negocio, es un wrapper a otros.

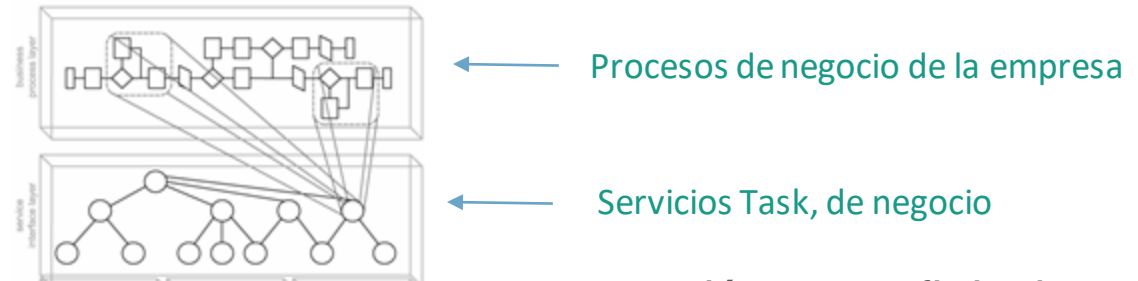
Arquitectura SOA- Detalle Clasificación de los Servicios

2) ORCHESTRATION SERVICE LAYER/NEGOCIO:

Representa la capa de negocio de servicios compuestos, SOA federada que se divide en las siguientes capas

- **Capa No Agnóstica**

- **Capa Task** : Es la capa que tiene la lógica de negocio del core, es la capa que generalmente se compone o reutiliza a la capa de entidades o su misma capa, por eso se denomina también capa de orquestación. Los servicios de esta capa realizan algo específico del negocio de la empresa:

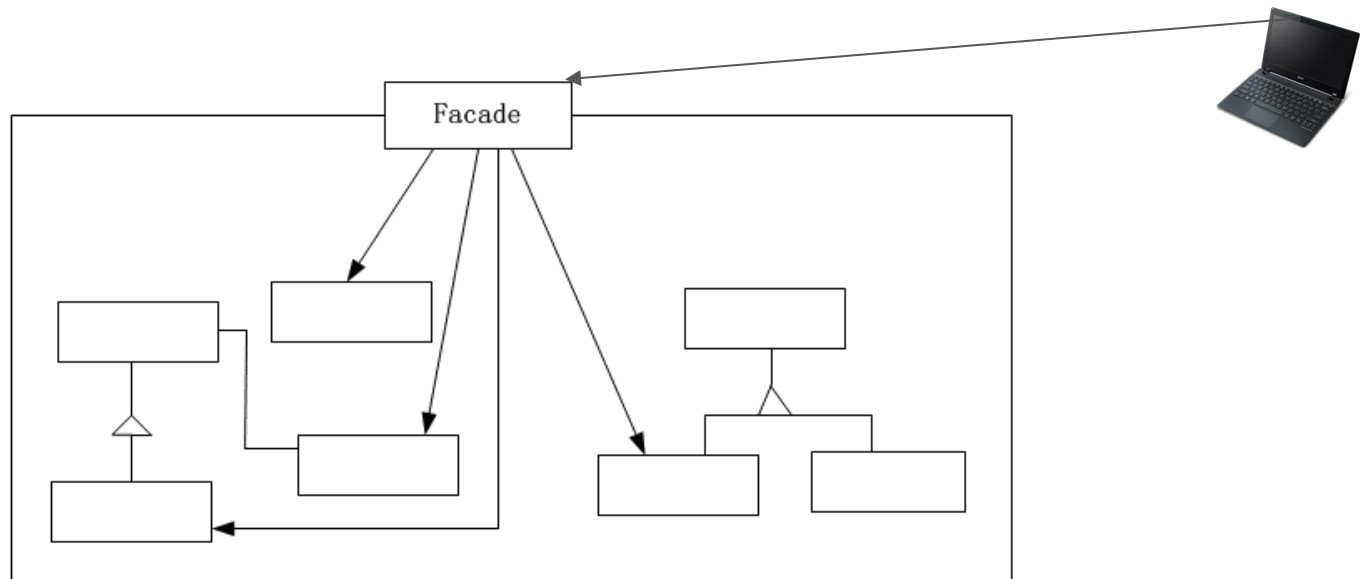


- **Capa BPEL**: También es una capa de orquestación, es un flujo de procesos donde está involucrado uno o más servicios pero con un factor de **tiempo significativo mayor al de un proceso síncrono o al anterior**. Ejemplo: evaluación crediticia, contratación de personal, representan muchas veces el mismo proceso de negocio de la empresa.

Arquitectura SOA- Detalle Clasificación de los Servicios

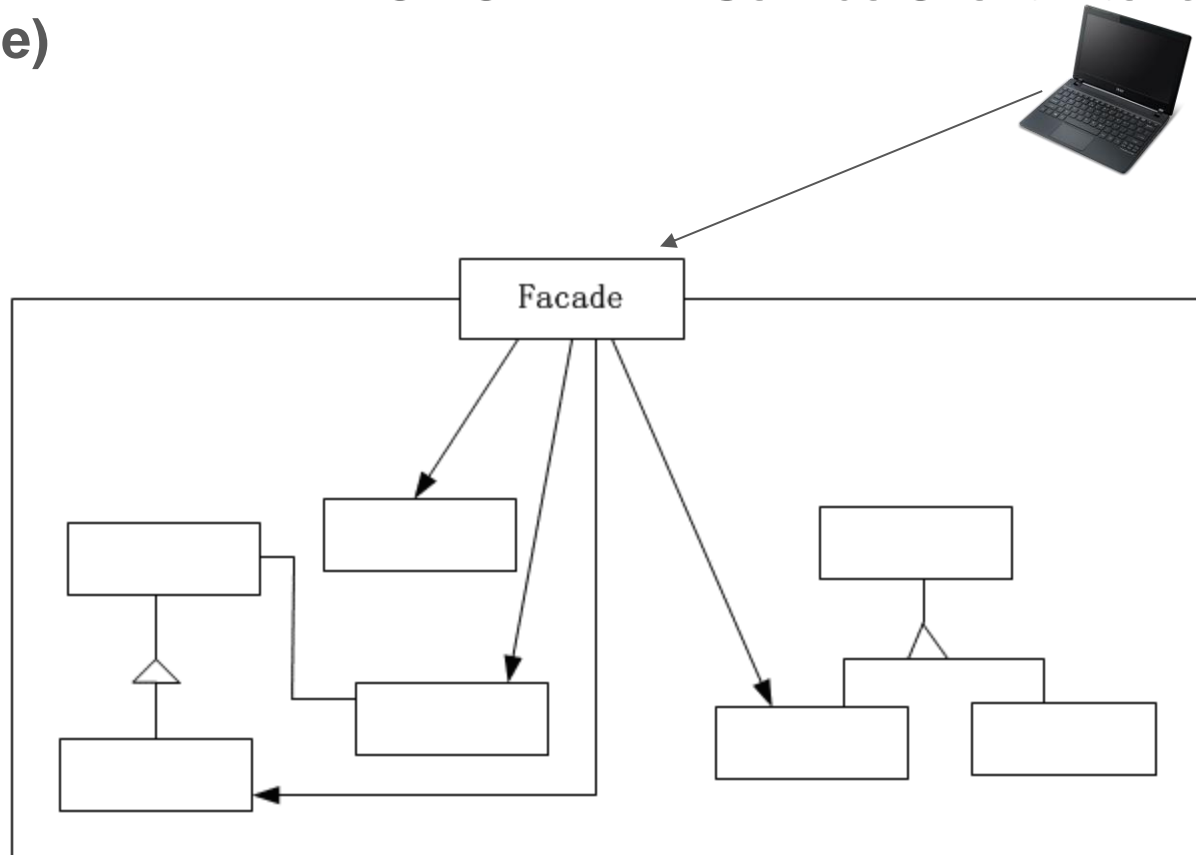
3) CAPA DE INTERFACE CLIENTE

- Sirve para **publicar o exponer algún servicio o lógica de la capa de orquestación hacia los clientes**, se le denomina **Service Client Interface** o simplemente **controladores** en el mundo de patrones de diseño como MVC.

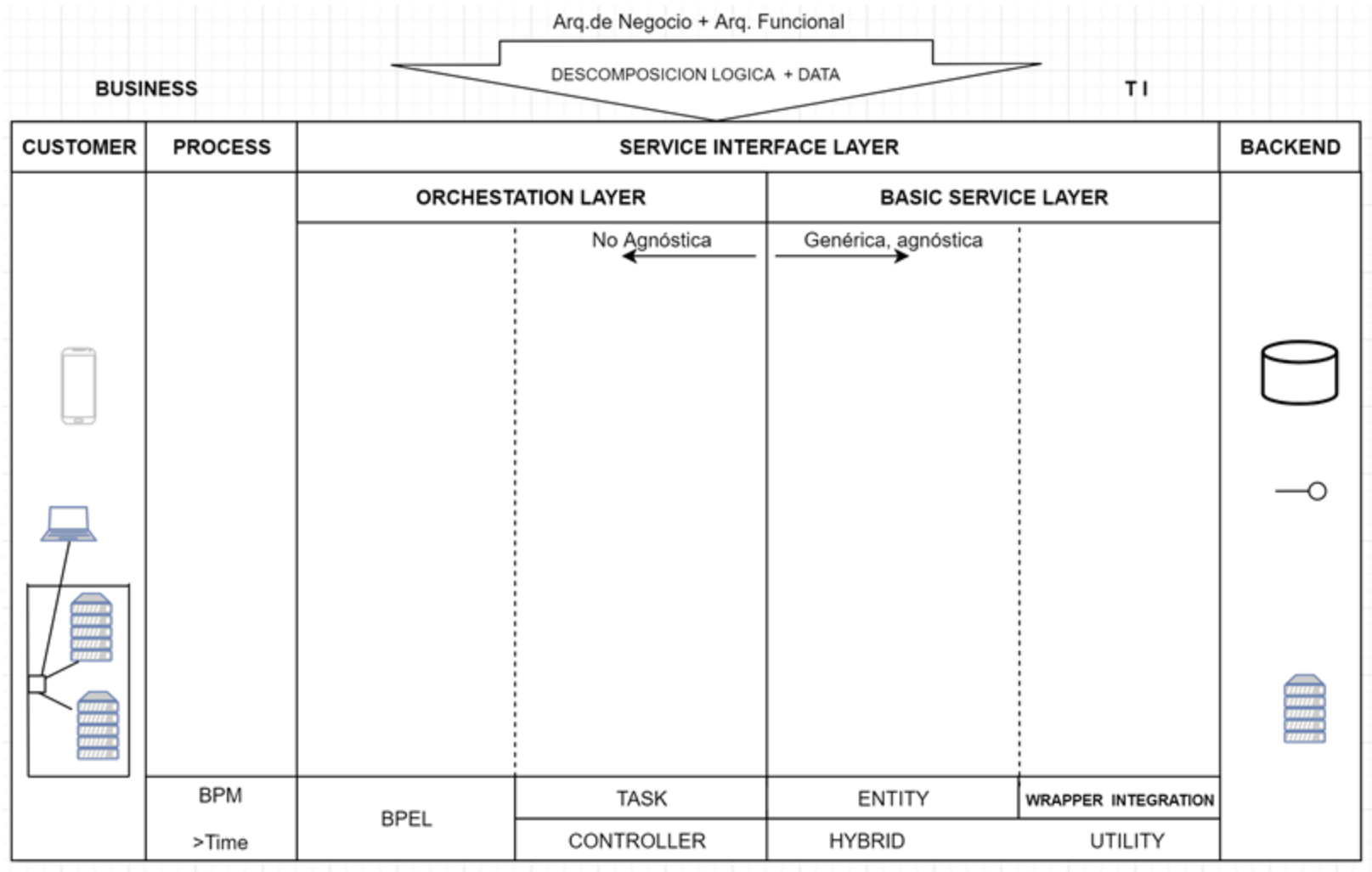


Arquitectura SOA- Detalle Clasificación de los Servicios

3) CAPA DE INTERFACE CLIENTE Service Client Interface (Facade)



Aquitectura SOA



Arquitectura de Servicios está compuesto de ...

➤ Services

- Unidades lógicas de procesamiento
- Ejemplo: Servicio de Tarjeta de Crédito

➤ Messages

- Unidades de comunicaciones entre servicios
- Necesario para que los servicios hagan su trabajo
-

➤ Operations

- Unidades de trabajo
- Example: Determinar el coste de atención

➤ Processes

- Compuesto / grupos de servicios orquestados
- Example: Evaluación Crediticia

Método para Identificar Servicios

1. Diseñar el diagrama de clases POO del Sistema

**2. Seleccionar los métodos de negocio y separar:
Entidades**

- Sólo entidades

Capa Agnóstica

- Funcionalidades DAO que trabajan con (Repositorio), incluye CRUD
- Funcionalidades que invocan servicios o recursos externos (Capa de Integración)
- Utilitarios (Capa de integración y Utilitarios)

3. Capa de Negocio u Orquestación, No Agnóstica

Funcionalidades que reutilizan servicios DAO o así mismos

4. Aplicar Principios SOA



Métodos para Identificar Servicios

Ejercicio

La Dirección de Tránsito de Lima necesita un programa que permita gestionar las multas por infracciones de tránsito de los conductores de Lima Metropolitana, así como la actualización de su puntaje. Para esto el sistema permitirá registrar los datos de los conductores de vehículos: DNI, número de licencia de conducir, nombre, dirección, email, año de registro y un puntaje de 300 puntos por defecto el cual indica que no tiene infracciones.

Las penalidades en puntaje por infracciones se aplican según:



Código Infracción	Descripción	Puntaje
A	Simple	-20
B	Básico	-50
C	Intermedio	-100
D	Alto	-200
E	Crítico	-300



La multa de infracciones al conductor se calcula de la siguiente manera:

$$0.20 * (\text{año de registro} - \text{año actual}) * (\text{puntaje actual}) \text{ en soles}$$

Para esta práctica no será necesario crear la tabla o entidad Penalidad.

Se pide:

- a) Registrar 3 conductores. (al finalizar cada registro debe enviarse un correo electrónico de bienvenida.
 - b) Dado un DNI obtener los datos de conductor.
 - c) Dado una licencia de conducir y código de penalidad actualice su récord de puntaje penalizándolo.
 - d) Un listado de todos los conductores cuyo vehículo tengan menos de 100 puntos.
 - e) Dado un número de DNI obtener la multa de infracción del conductor.
 - f) Invocar un servicio externo de <https://currencylayer.com/> para obtener una lista de tipos de cambio de un dólar en todas las monedas extranjeras y peruana.
-
- g) Diseñe su arquitectura SOA incluya sus entradas y salidas de todos los servicios, mencione los principios SOA que aplica en su diseño/programa.

Adjunte sus pantallazos de resultados de pruebas unitarias y base de datos por cada prueba, también su proyecto con código fuente e imagen del diagrama SOA.

Administrador

+arregloConductores<>

+**registrar(conductor)
+**buscar_conducto(dni)
+** buscar_conductor(codigo)
+**obtener_conductores_100()
+**actualizar_conductor(conductor)
+** obtener_datos_conductor(dni)
+obtener_pago_anual_impuesto(dni)
+enviarEmail(dni)
+obtenerTiposCambio()

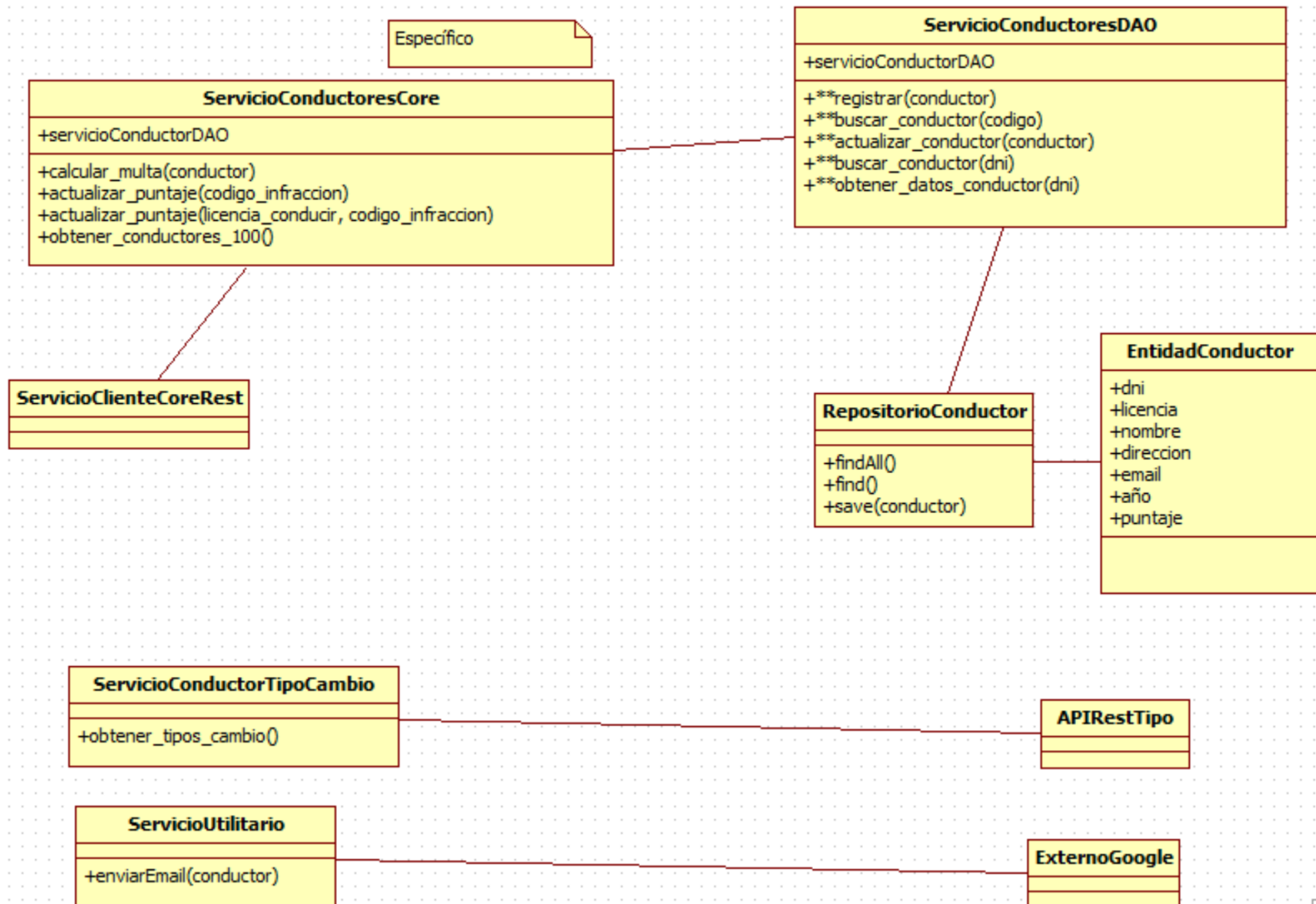
1..*

Conductor

+dni
+licencia
+nombre
+direccion
+email
+año
+puntaje

+calcular_multa()
+enviarEmail()
+actualizar_puntaje(codigo_infraccion)

Específico

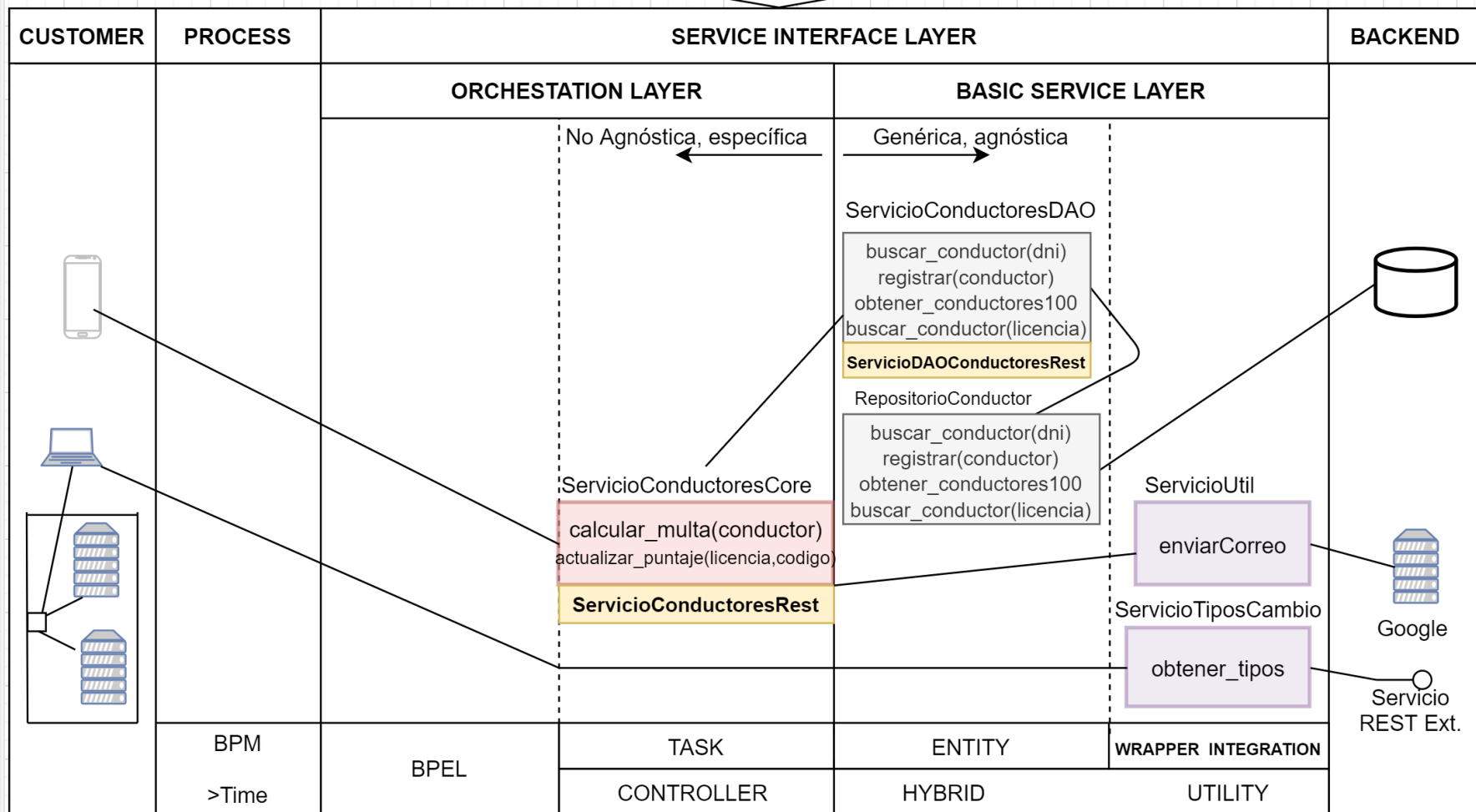


Arq.de Negocio + Arq. Funcional

DESCOMPOSICION LOGICA + DATA

BUSINESS

TI




Otro método de Diseño SOA

- Descomposición de Procesos en Servicios
(ver video del Aula Virtual y Referencias)

Referencias

- ISO 12207 (<http://www.12207.com/>).
- The Open Group Architecture Framework (<http://www.opengroup.org/>).
- OASIS Reference Model for Service Oriented Architecture (http://www.oasis-open.org/committees/tc_cat.php?cat=soa).
- Enterprise SOA: Service-Oriented Architecture Best Practices. Dirk Krafzig, Karl Banke, Dirk Slama. Prentice Hall PTR. November 09, 2004. ISBN 0131465759.
- IBM SOA (<http://www-306.ibm.com/software/solutions/soa/>).
- Oracle SOA (<http://www.oracle.com/technologies/soa/index.html>).
- Identificación de servicios por descomposición
https://www.youtube.com/watch?feature=player_detailpage&v=TauAUHGtH_w#t=433
- Arquitectura
<https://youtu.be/y46TBEf21Gs>
- https://www.ted.com/talks/ole_scheeren_why_great_architecture_should_tell_a_story/transcript?language=es



Principios de orientación a servicio Principios SOA

Principios de Orientación a Servicios

1. Contrato Estandarizado
2. Bajo Acoplamiento
3. Abstracción
4. Reutilización
5. Autonomía
6. Sin Estado
7. Descubrimiento
8. Composición

Thomas Earl -2,005

Objetivos organizacionales:

- ✓ Aumentar la interoperabilidad intrínseca (intercambio de información natural, sin transformaciones)
- ✓ Aumentar la federación (gobierno menos centralizado)
- ✓ Aumentar las alternativas de diversificación de fabricantes o proveedores (heterogeneidad)
- ✓ Aumentar el alineamiento negocio-tecnología (servicios orientados al negocio)
- ✓ Aumentar el ROI (mejor retorno de inversión)
- ✓ Aumentar la agilidad organizacional (soluciones transversales, capacidad de respuesta a los cambios, moverse al mismo ritmo)
- ✓ Reducir la carga de TI (menos retrabajo y mantenimiento, no silos, la empresa en su conjunto se simplifica)

Principio 1: Contrato estandarizado

Contrato de servicio: Corresponde a la parte pública de un servicio; incluye su descriptor ¿qué ofrece?, ¿dónde se ubica?, ¿cómo invocarlo?, esquema(s), políticas y acuerdos de nivel de servicio.

“Los servicios dentro del mismo inventario o dominio siguen los mismos **estándares** de diseño de contrato.”

Áreas de estandarización:

1. Expresiones funcionales.- Nomenclatura y tipos de mensajes.
2. Modelo de datos.- Estructuras y tipos de datos.
3. Políticas.- Requerimientos y términos de uso. SLA o nivel de servicio que un cliente espera de su proveedor.



WSDL – XSL

Doc. de diseño

SLA

WSLD

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://app.com/productos" targetNamespace="http://app.com/productos">
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
      <xs:element name="GetProductoRequest">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="id" type="xs:int"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="GetProductoResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="Producto" type="tns:Producto"/>
            <xs:element name="codigoRespuesta" type="xs:int"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:complexType name="Producto">
        <xs:sequence>
          <xs:element name="codigo" type="xs:int"/>
          <xs:element name="descripcion" type="xs:string"/>
          <xs:element name="precio" type="xs:double"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
</wsdl:definitions>
```

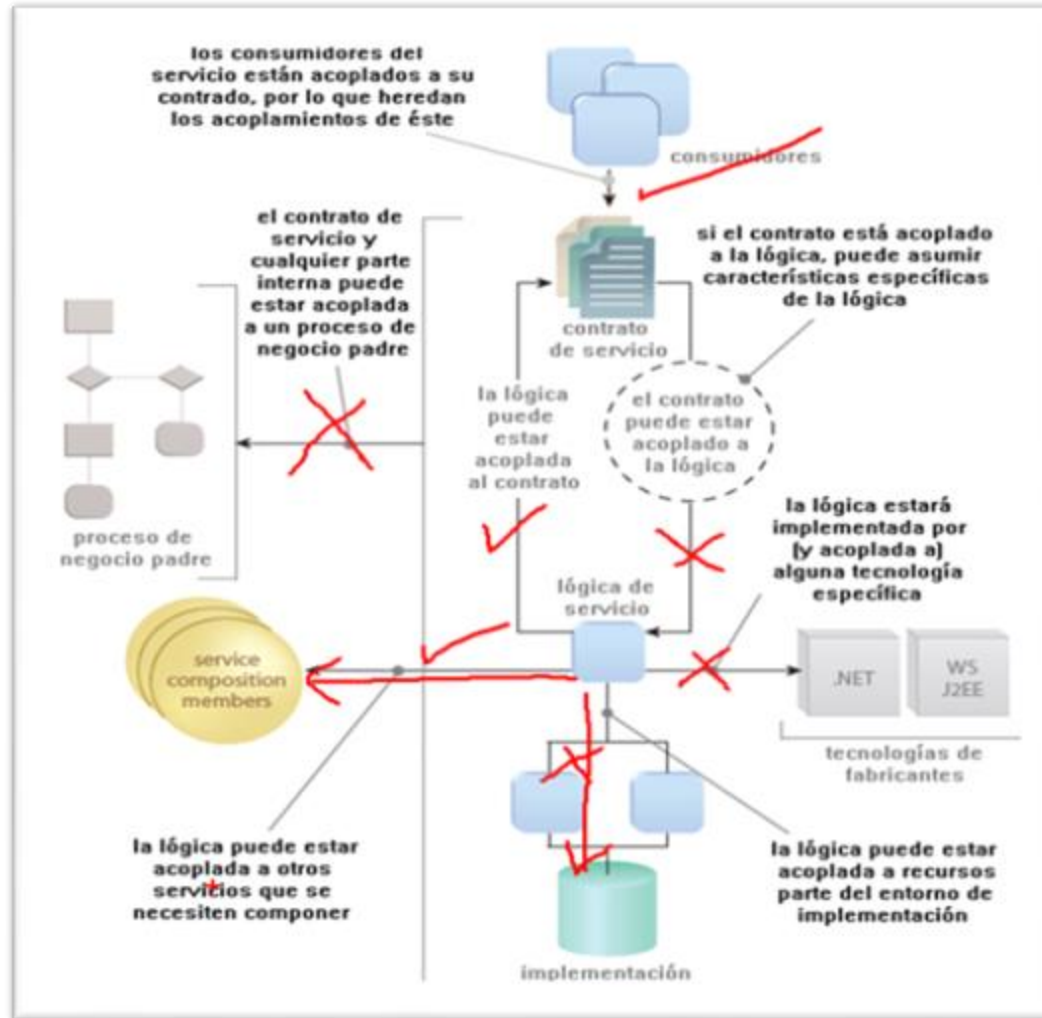
```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:prod="http://app.com/productos">
  <soapenv:Header/>
  <soapenv:Body>
    <prod:GetProductoRequest>
      <prod:id>102</prod:id>
    </prod:GetProductoRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:GetProductoResponse
xmlns:ns2="http://app.com/productos">
      <ns2:Producto>
        <ns2:codigo>102</ns2:codigo>
        <ns2:descripcion>Lapiz</ns2:descripcion>
        <ns2:precio>1.0</ns2:precio>
      </ns2:Producto>
      <ns2:codigoRespuesta>1</ns2:codigoRespuesta>
    </ns2:GetProductoResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<wsdl:message name="GetProductoRequest">
  <wsdl:part element="tns:GetProductoRequest" name="GetProductoRequest"> </wsdl:part>
</wsdl:message>
<wsdl:message name="GetProductoResponse">
  <wsdl:part element="tns:GetProductoResponse" name="GetProductoResponse"> </wsdl:part>
</wsdl:message>
<wsdl:portType name="ProductoPort">
  ▼<wsdl:operation name="GetProducto">
    <wsdl:input message="tns:GetProductoRequest" name="GetProductoRequest"> </wsdl:input>
    <wsdl:output message="tns:GetProductoResponse" name="GetProductoResponse"> </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ProductoPortSoap11" type="tns:ProductoPort">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  ▼<wsdl:operation name="GetProducto">
    <soap:operation soapAction=""/>
    ▼<wsdl:input name="GetProductoRequest">
      <soap:body use="literal"/>
    </wsdl:input>
    ▼<wsdl:output name="GetProductoResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
▼<wsdl:service name="ProductoPortService">
  ▼<wsdl:port binding="tns:ProductoPortSoap11" name="ProductoPortSoap11">
    <soap:address location="http://localhost:8080/ws"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Principio 2: Bajo acoplamiento

“El objetivo del bajo acoplamiento es **minimizar las dependencias**”

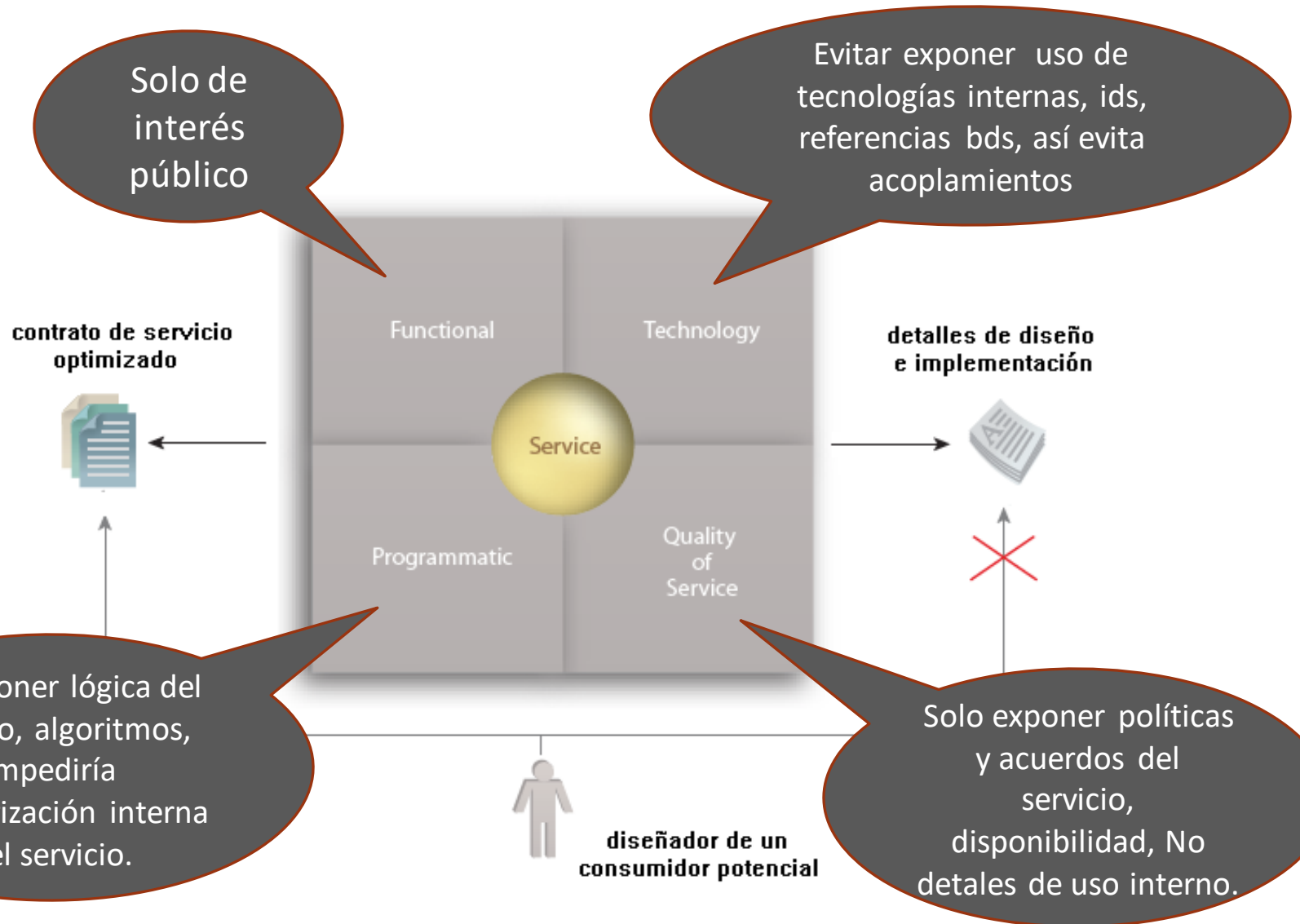


Por que es importante un bajo acoplamiento?

	Alto Acoplamiento	Bajo acoplamiento
Conexiones físicas	Punto a punto	A través de mediador
Estilo de comunicación	Sincrónico	Asincrónico
Modelo de datos	Tipos complejos comunes	Tipos comunes simples solamente
Tipo de sistema	Fuerte	Débiles
Control de la lógica del proceso	Control central	Control distribuido
Unión / Binding	Inactivamente	Dinámicamente
Plataforma	Fuertes dependencias de plataforma	Plataforma independiente
Transaccionalidad	2PC (de dos fases)	Compensación
Despliegue	Simultáneo	En Diferentes Momentos
Versiones	Actualizaciones explícitas	Actualizaciones implícitas

Principio 3: Abstracción.

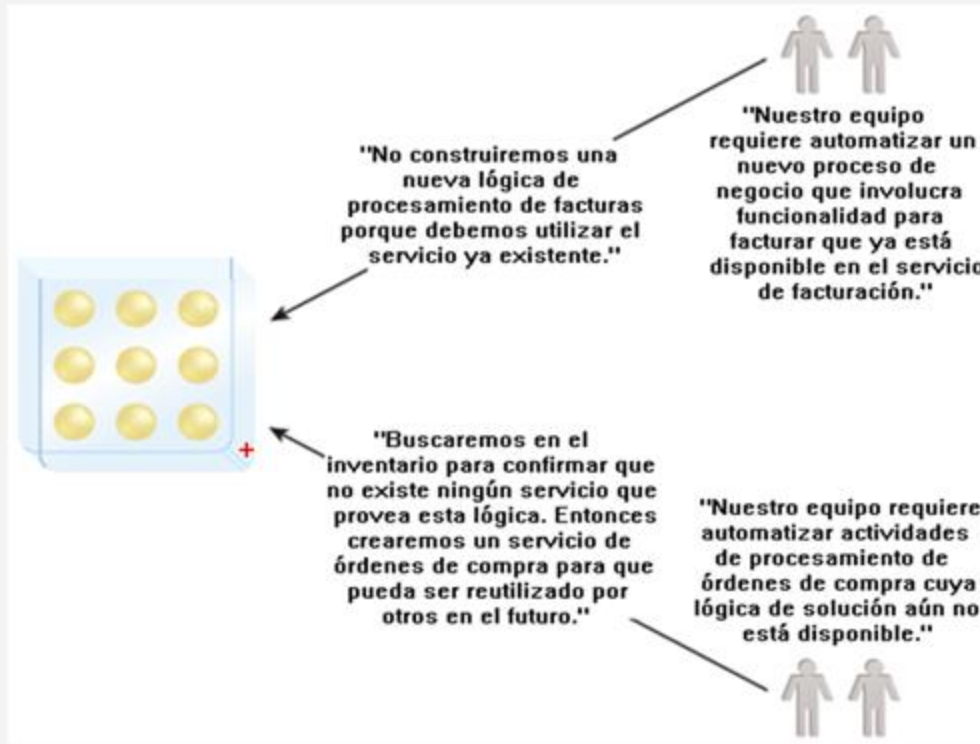
“Los contratos de servicios **exponen sólo la información esencial** acerca del servicio.”



Principio 4: Reutilización

“Los servicios contienen y expresan lógica que puede ser aprovechada como recurso **reutilizable** para la organización desde el punto de vista de **negocio**.”

"Potencial... de Reutilización"



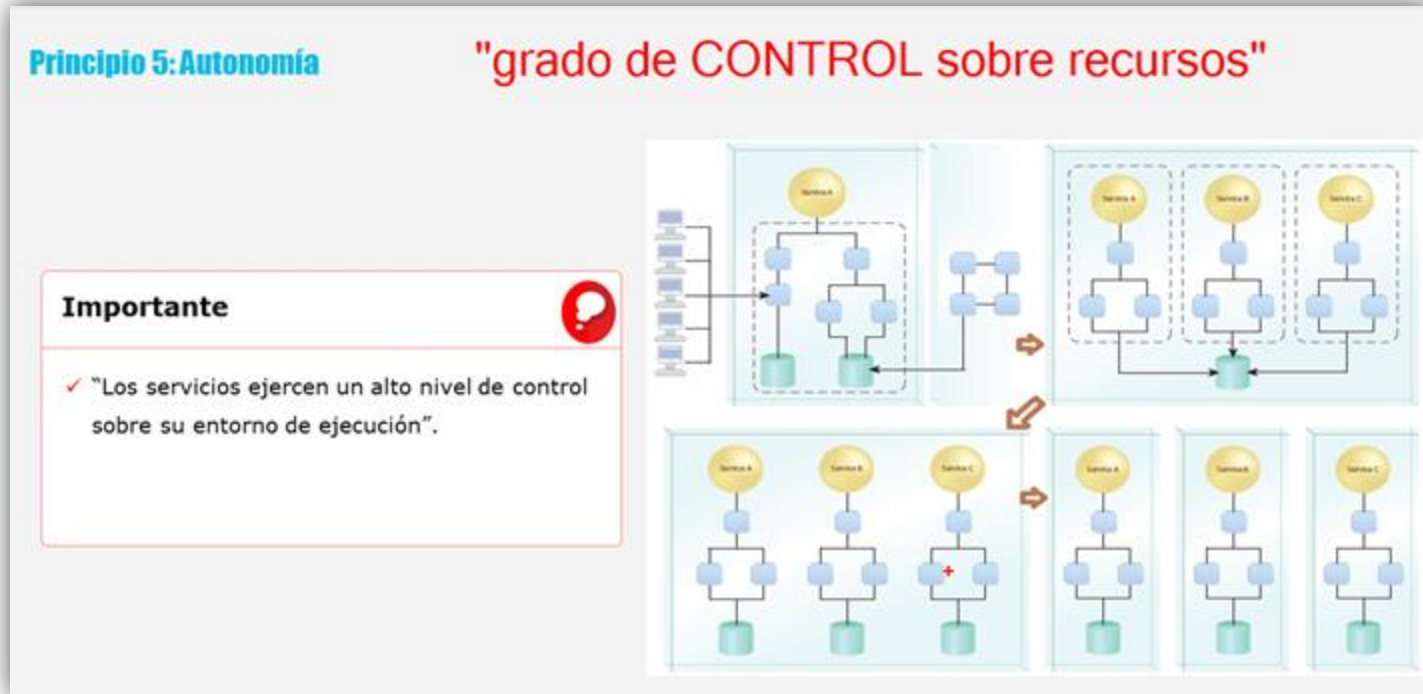
General-particular

Buscar antes de hacer

Una impresora en red con su driver, es SOA?

Principio 5: Autonomía

El servicio tiene un alto grado de control sobre su entorno de ejecución y sobre la lógica que encapsula.



Un servicio es autónomo porque su única relación con el mundo exterior, al menos desde la perspectiva de SOA, es a través de su interfaz o contrato.

Autogobierno, autocontrolado [Auto-self].

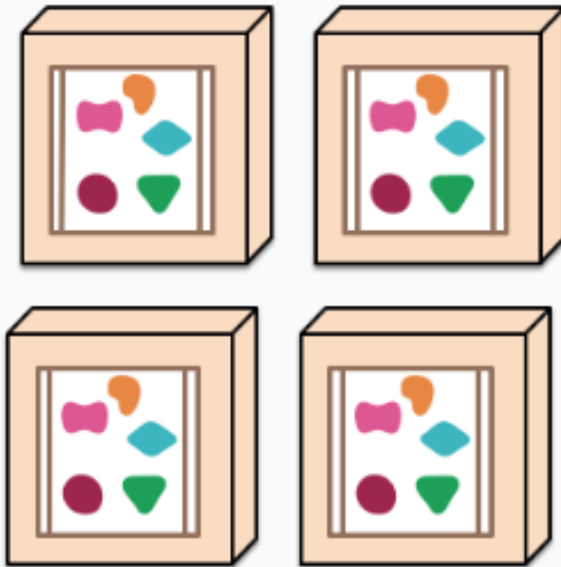
Un servicio no puede contener lógica que dependa de nada externo al propio servicio, ya sea un modelo de datos, un sistema de información, o cualquier otra cosa, acoplamiento cero=microservicio.

Una arquitectura de microservicios pone cada elemento de funcionalidad en un servicio separado y escala redistribuyendo estos servicios a través de los servidores, replicándolos según se vaya necesitando.

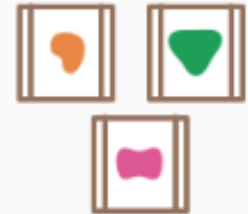
A monolithic application puts all its functionality into a single process...



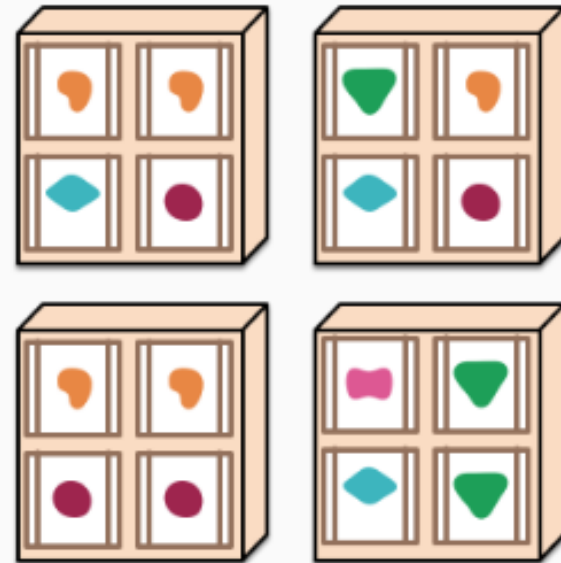
... and scales by replicating the monolith on multiple servers



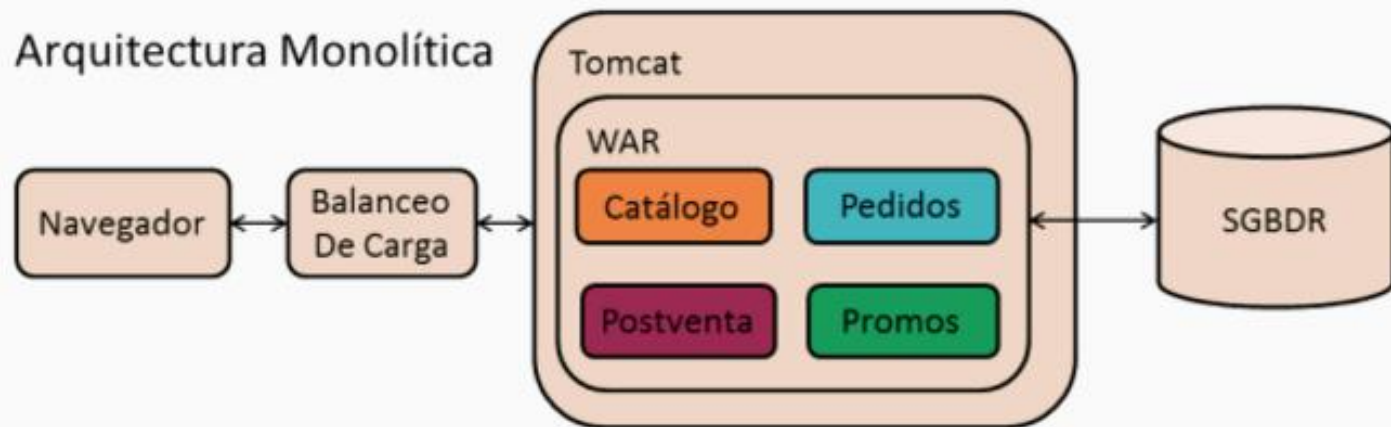
A microservices architecture puts each element of functionality into a separate service...



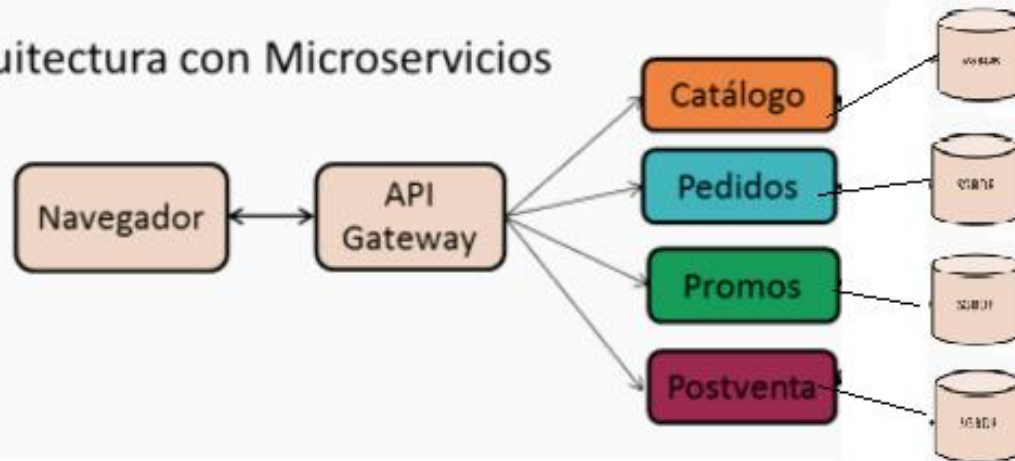
... and scales by distributing these services across servers, replicating as needed.



Arquitectura Monolítica



Arquitectura con Microservicios

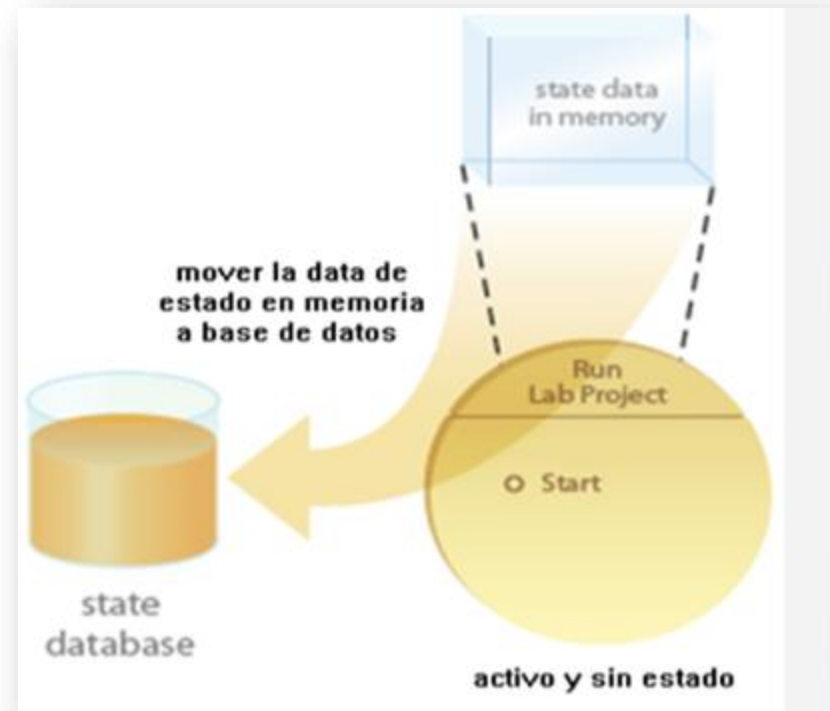


Principio 6: Sin estado

Alzhéimer

Los servicios SOA no deben guardar información alguna sobre datos de sesión, ni sobre eventos **previos**, ni sobre resultados de servicios invocados previamente. Es decir, **los servicios no deben tener estado**.

- “Los **servicios minimizan el uso** de recursos de manejo de estado y de ser necesario lo delegan para mejorar su **escalabilidad**.”
- Idealmente, todos los datos que necesita el servicio para trabajar deben provenir directamente de los parámetros de entrada.
- Ideal, requisito para la **composición**.
- Incrementa la **escalabilidad**.
- Mejora la **autonomía**.
- Total **disponibilidad**
- **Mejor rendimiento**



“**corren buenos tiempos para el desarrollador de backend**, sólo tiene que preocuparse por una invocación al servicio y dar una respuesta. El resto se lo tienen que “comer” los desarrolladores de frontend?”

Principio 7: Descubrimiento

“Los servicios son descritos con información de sus capacidades para que puedan ser utilizados y entendidos.”

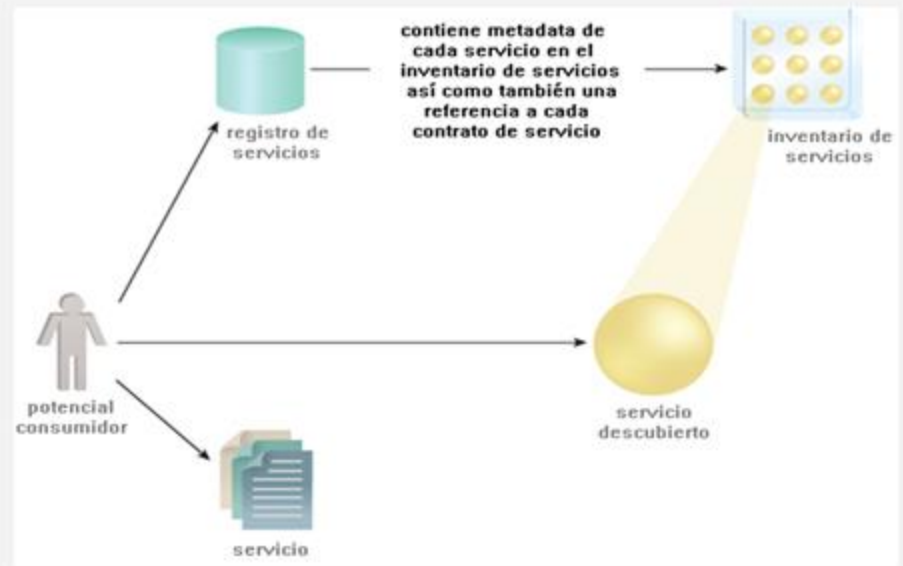
Principio 7: Descubrimiento

-publicar } servicios
-buscar }

Importante

- ✓ “Los servicios son descritos con información de sus capacidades para que puedan ser ubicados y entendidos”.

"Catálogo de servicios"



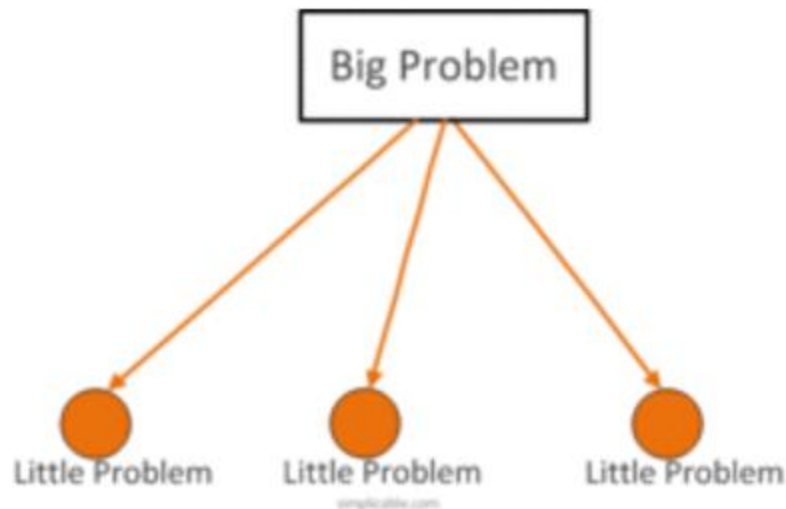
ws-discovery

Principio 8: Facilidad de Composición

Este principio nos dice que todo servicio debe estar construido de tal manera que pueda ser utilizado para construir servicios genéricos de mayor complejidad de mayor jerarquía reutilizable general.

Empatía

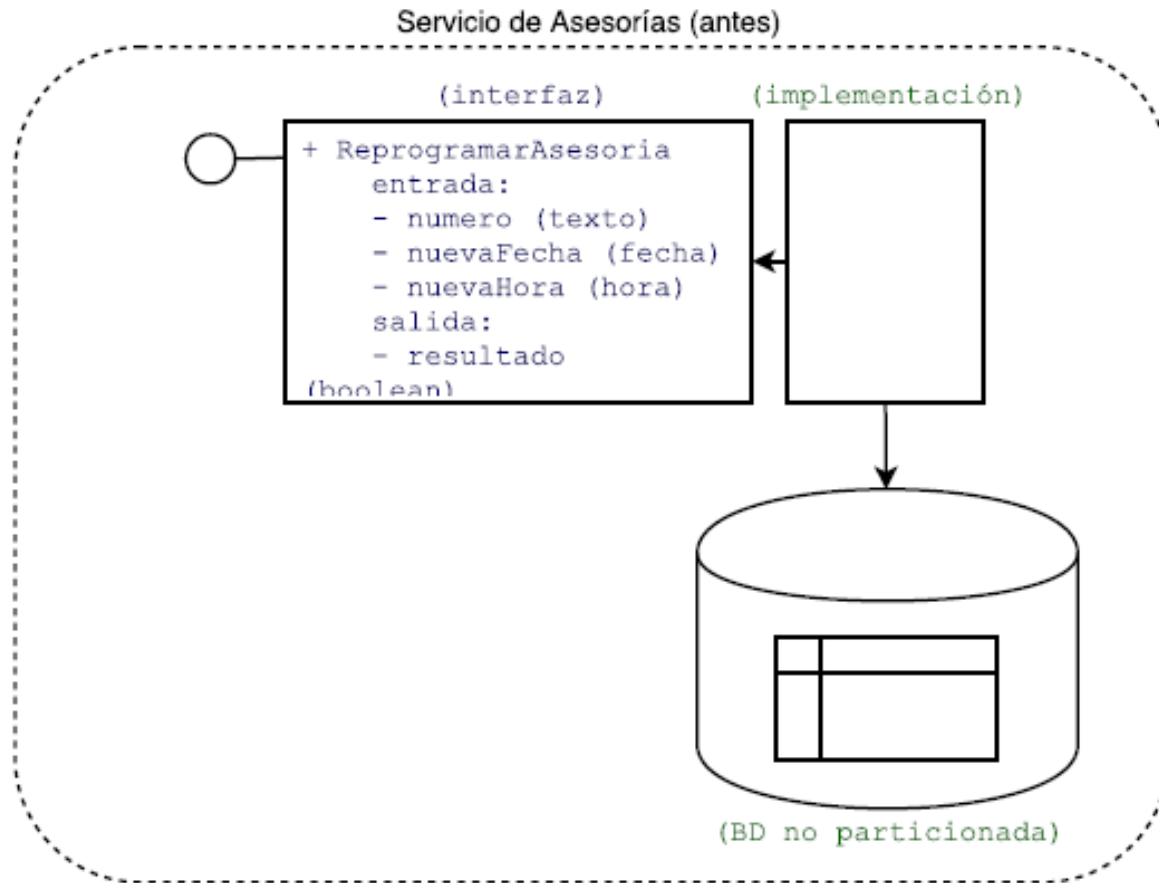
Componer diferentes capacidades de negocio.



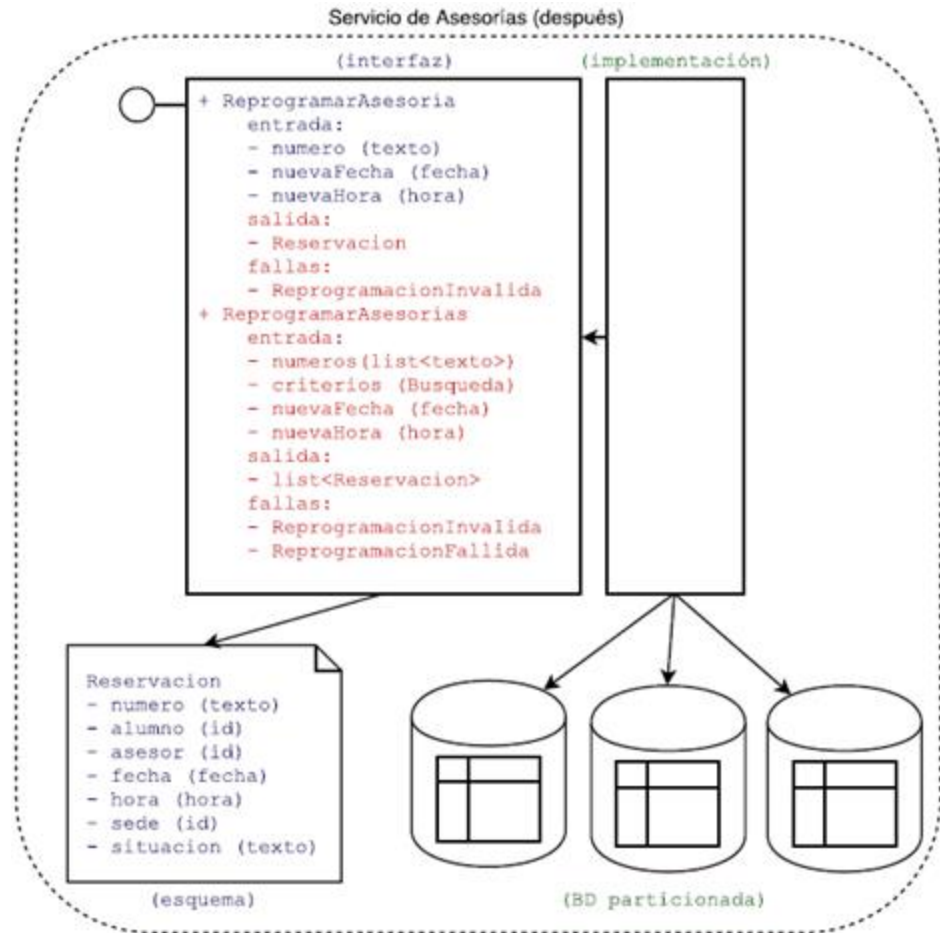
Composición: Automatización de un proceso o tarea de negocio.

Cuando no existe ningún servicio que satisfaga las funcionalidades requeridas por el solicitante, debe existir la posibilidad de combinar varios servicios existentes que en conjunto cumplan con la necesidad deseada

Caso: Un servicio con poca capacidad de ser compuesto



Solución 8: Un servicio con mayor capacidad de ser compuesto



Resumen

