



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.

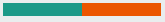
- Servicios

- 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.





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

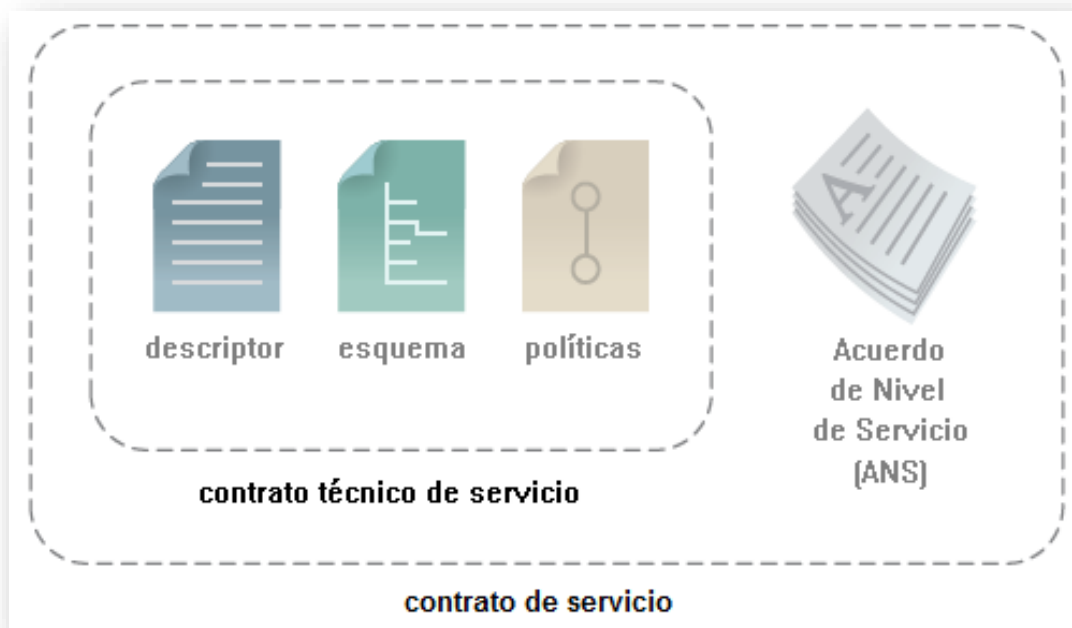
Objetivos organizacionales:

- ✓ 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:

“Todos los Contratos de Servicios del Catálogo de Servicios SOA deben cumplir los mismos estándares y normas de diseño”.



WSDL – XSL

SLA

Documento de diseño

Principio 1: Contrato estandarizado

Áreas de estandarización:

- **Estandarización de la expresión funcional**

Las operaciones de servicios deben definirse como nomenclatura estandarizada con nombres de entrada y salidas de mensajes y sus tipos.

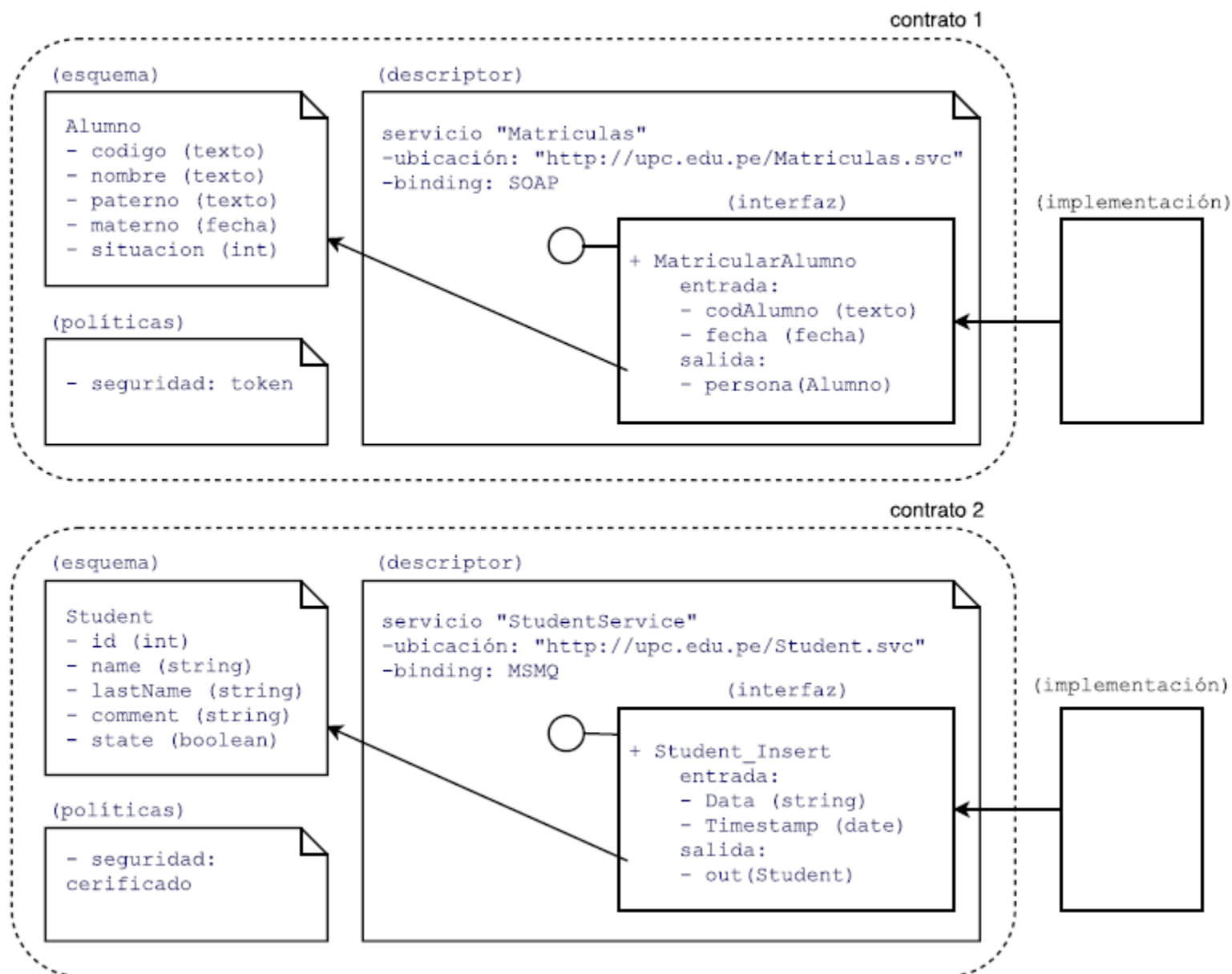
- **Estandarización del modelo de datos**

Estructuras y tipos de datos.

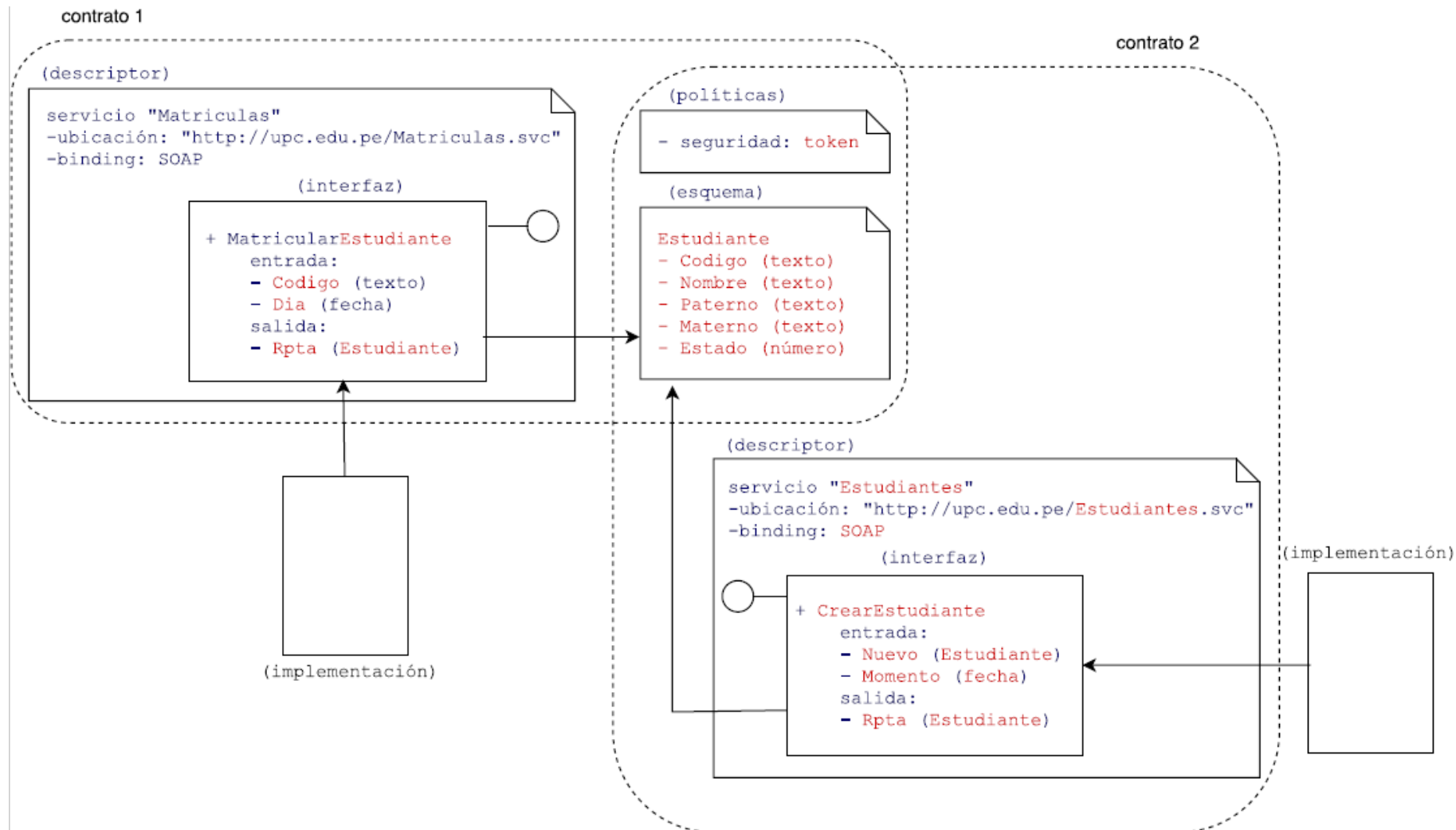
- **Estandarización de políticas**

Términos de uso de un servicio. SLA o nivel de servicio que un cliente espera de su proveedor basadas en estándares de la industria.

Caso 1: Dos servicios no estandarizados



Solución 1: Dos servicios estandarizados



Contrato Estandarizado

¿qué ofrece?, ¿dónde se ubica?, ¿cómo invocarlo?,
esquema(s), políticas y acuerdos de nivel de servicio.

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>

```

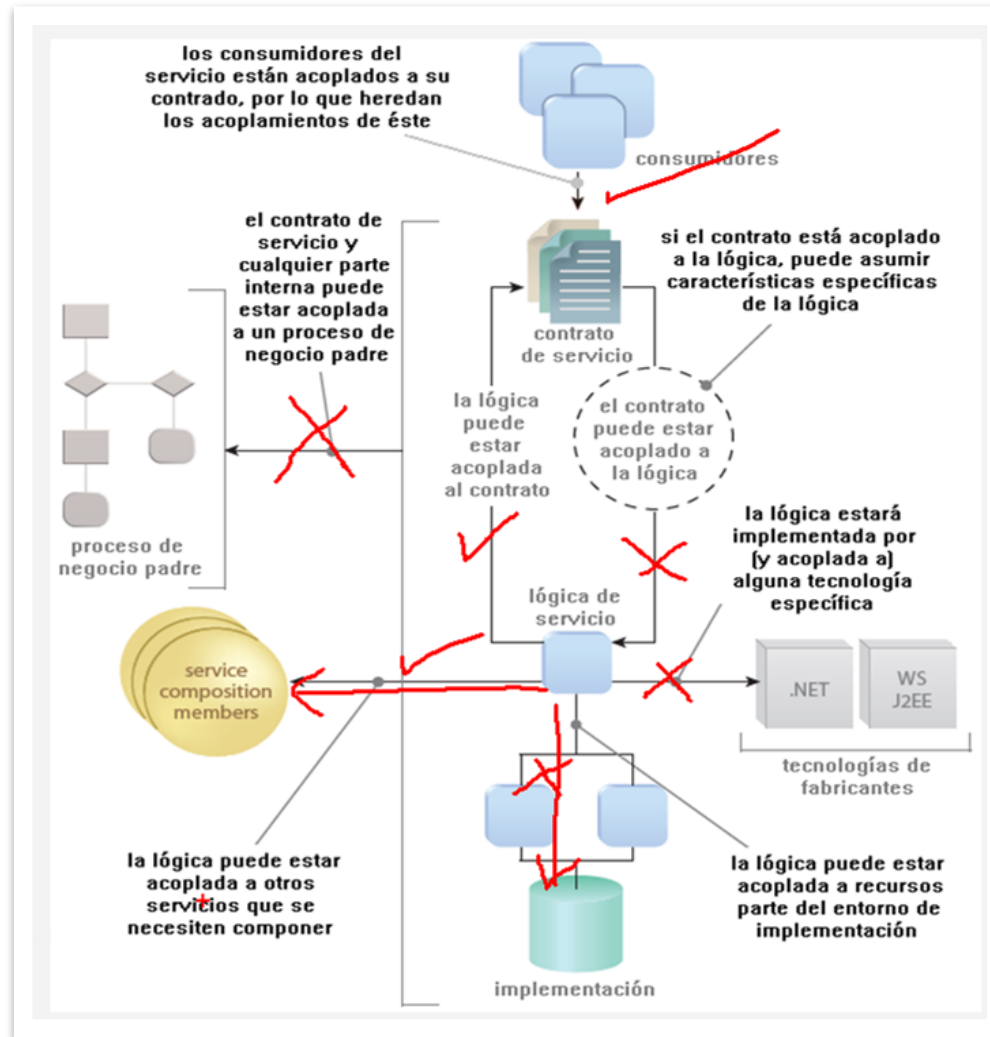
```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:prod="http://app.com/productos">
  <soapenv:Header/>
  <soapenv:Body>
    <prod:GetProductoRequest>
      <prod:id>102</prod:id>
    </prod:GetProductoRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

```
<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*”



V : Alto Acoplamiento

X: Bajo Acoplamiento

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
Patrón de interacción	Navegar a través de árboles de objetos complejos	Centrado en los datos, mensaje autocontenido
Control de la lógica del proceso.	Control central	Control distribuido
Unión	Inactivamente	Dinámicamente
Plataforma	Fuertes dependencias de plataforma	Plataforma independiente
Transaccionalidad	2PC (comillas de dos fases)	Compensación
Despliegue	Simultáneo	En Diferentes Momentos
Versiones	Actualizaciones explícitas	Actualizaciones implícitas

Alta Cohesión Bajo Acoplamiento

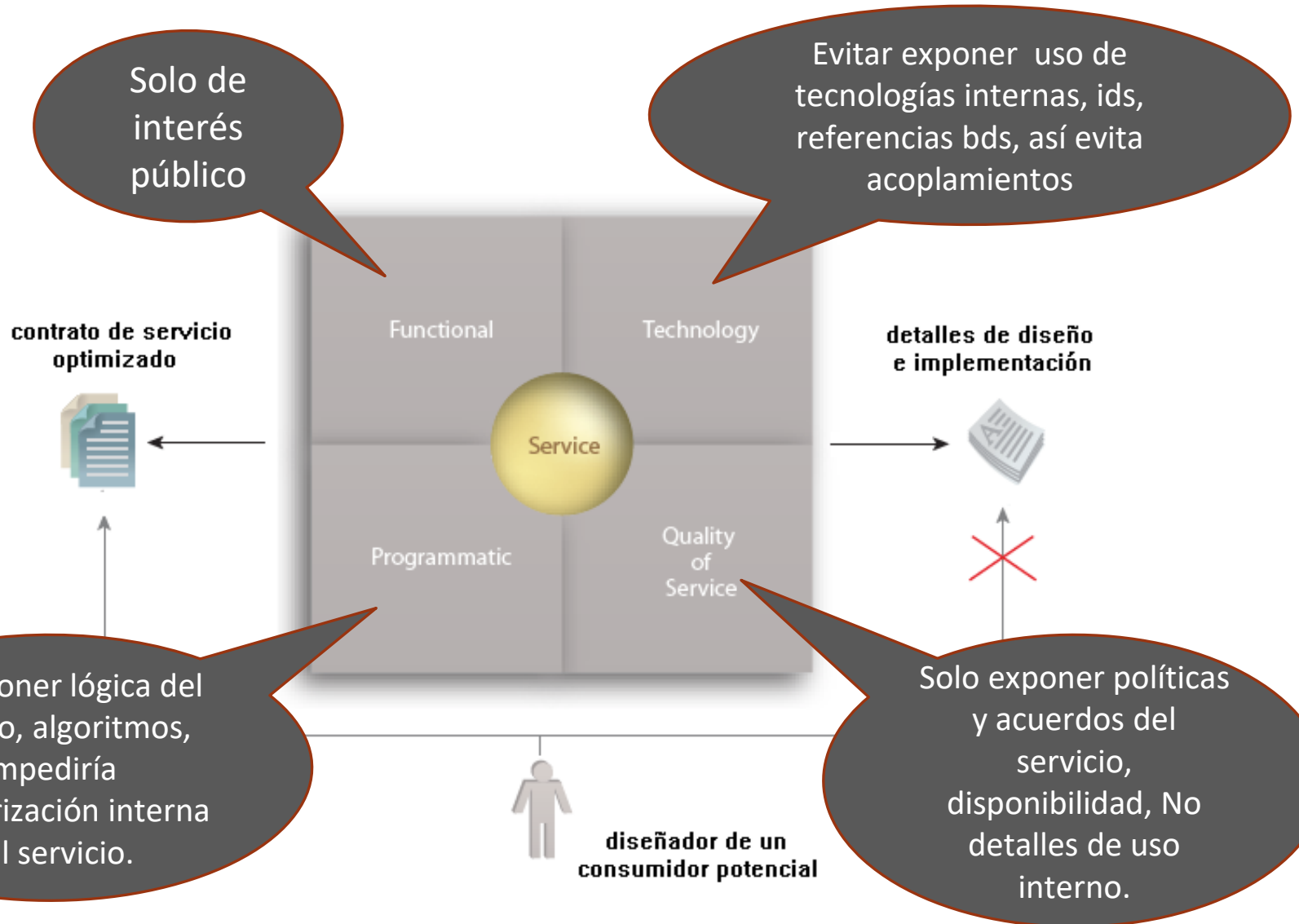
La cohesión es el grado en que los elementos de un determinado módulo pertenecen juntos, mientras que el acoplamiento es el grado en que un elemento conoce el funcionamiento interno de otro.

El principio de responsabilidad única de Robert C. Martin es una forma útil de considerar el primero:

*Reúna las cosas que cambian por las mismas razones.
Separe las cosas que cambian por diferentes razones.*

Principio 3: Abstracción.

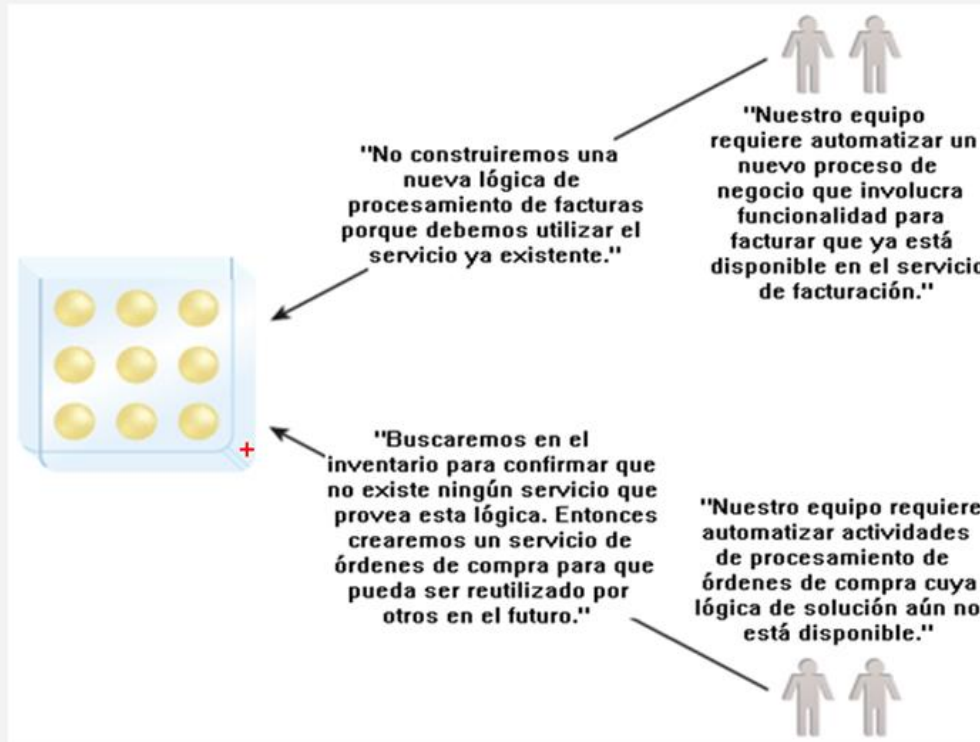
“Los contratos de servicios exponen sólo la información **esencial** acerca del servicio.”



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

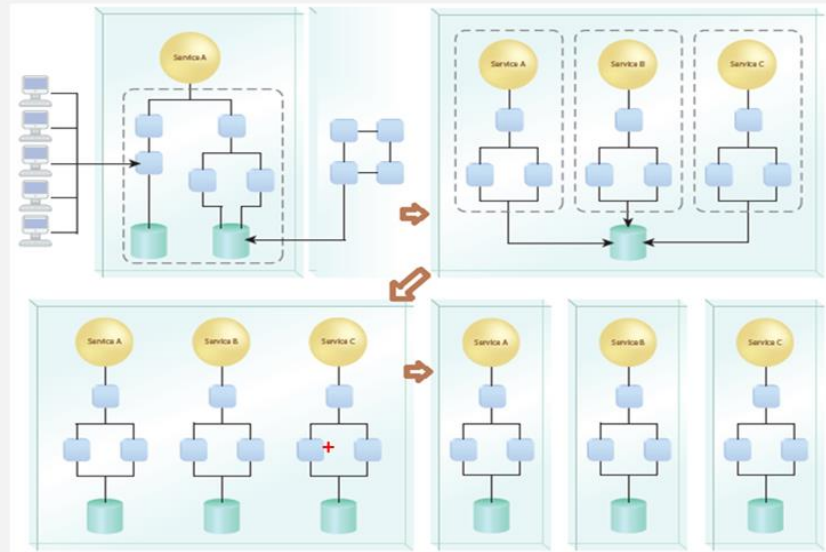
El servicio tiene un alto grado de control sobre su entorno de ejecución y sobre la lógica que encapsula.

Principio 5: Autonomía

"grado de CONTROL sobre recursos"

Importante

- ✓ "Los servicios ejercen un alto nivel de control sobre su entorno de ejecución".



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].

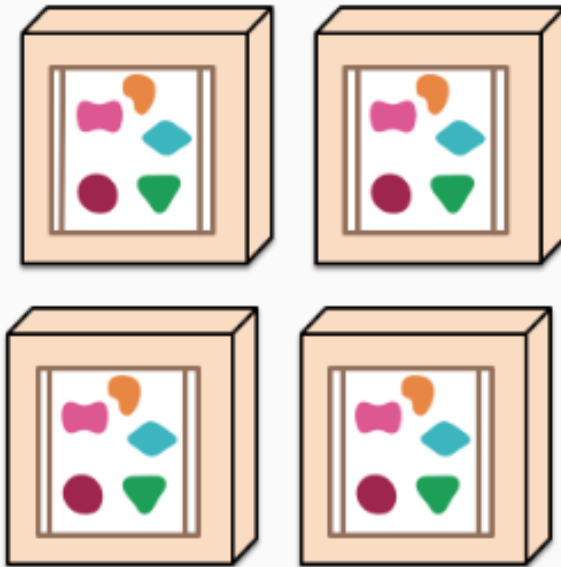
Un servicio no puede contener lógica que dependa de nada externo al propio servicio, ya sea un modelo de datos, un sistema de información, o cualquier otra cosa, acoplamiento cero=microservicio.

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...



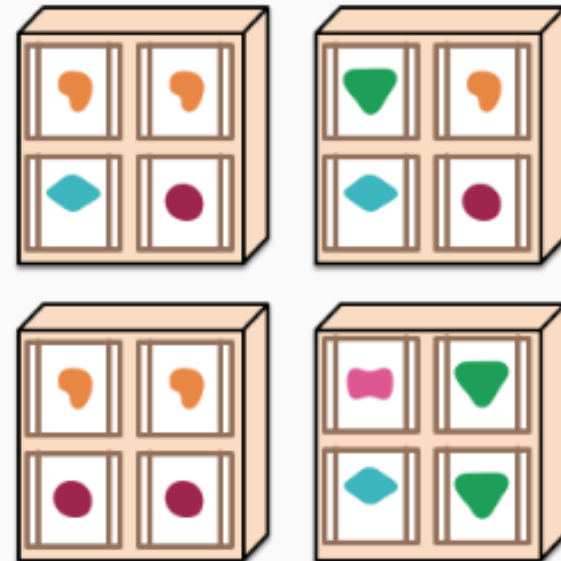
... and scales by replicating the monolith on multiple servers



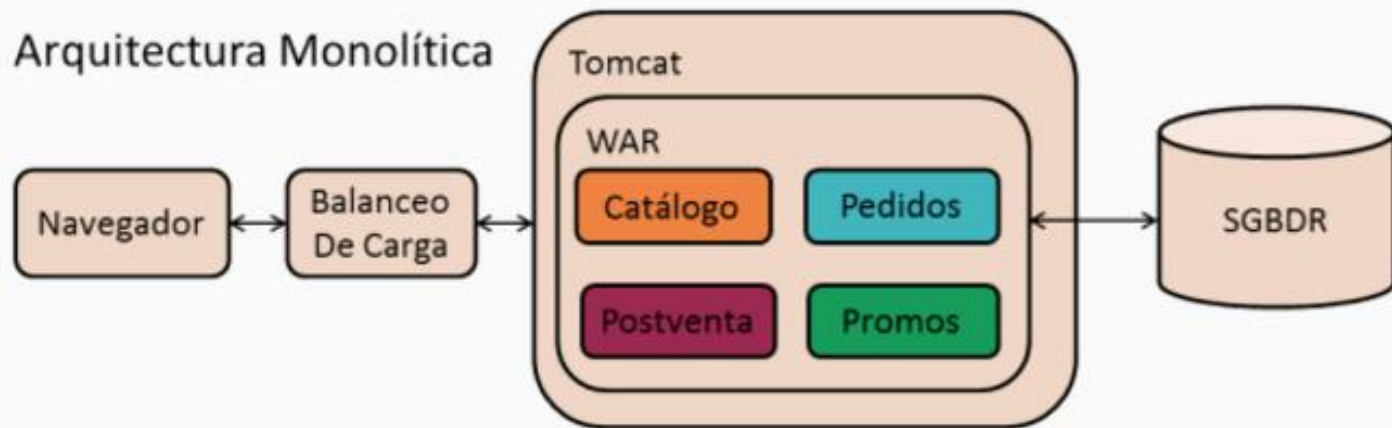
A microservices architecture puts each element of functionality into a separate service...



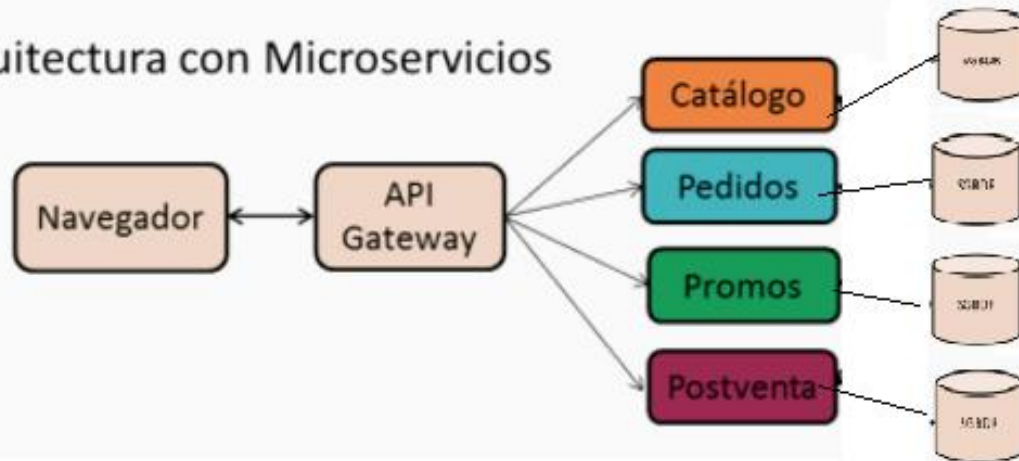
... and scales by distributing these services across servers, replicating as needed.



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**.

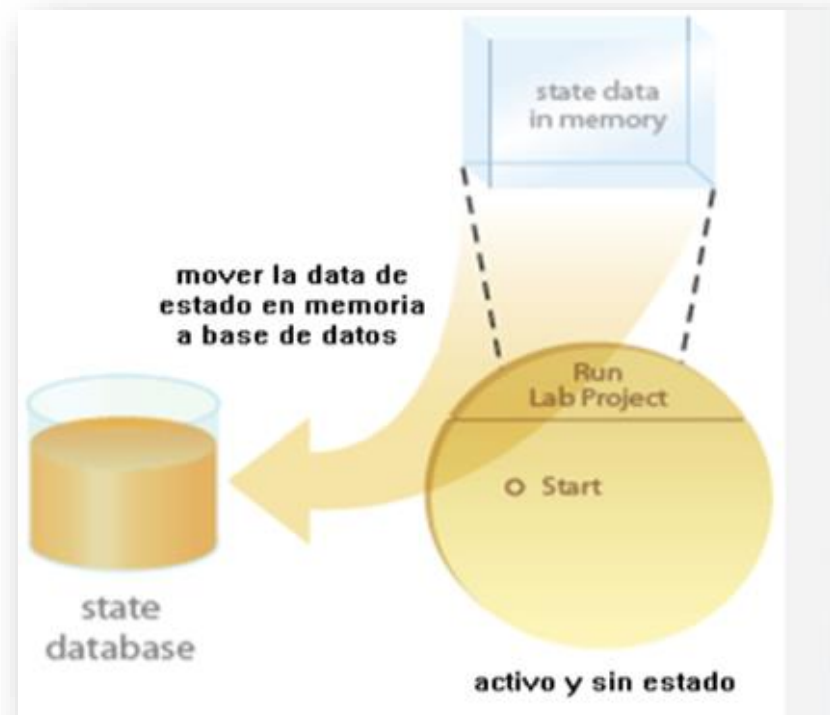
Incrementa la **escalabilidad**.

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



Principio 7: Descubrimiento

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

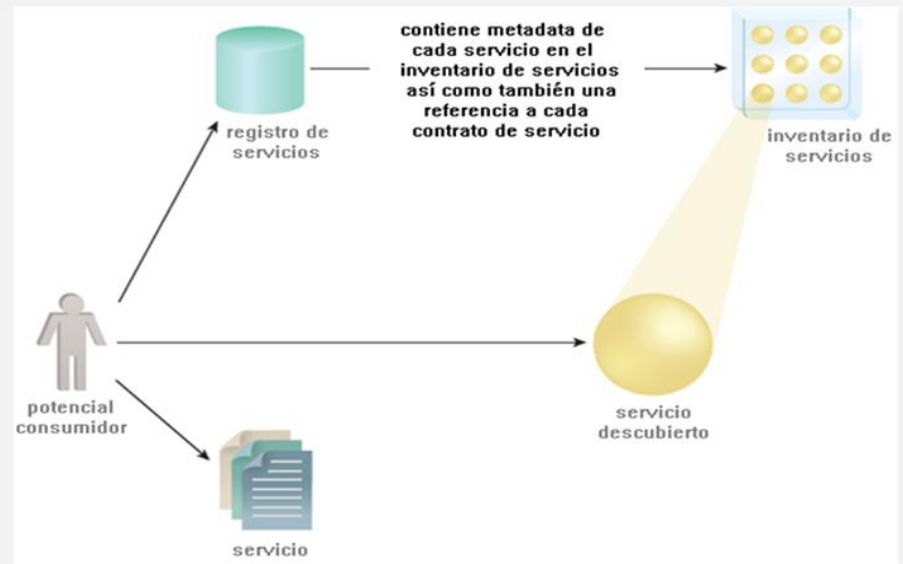
Principio 7: Descubrimiento

-publicar } servicios
-buscar }

Importante

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

"Catálogo de servicios"



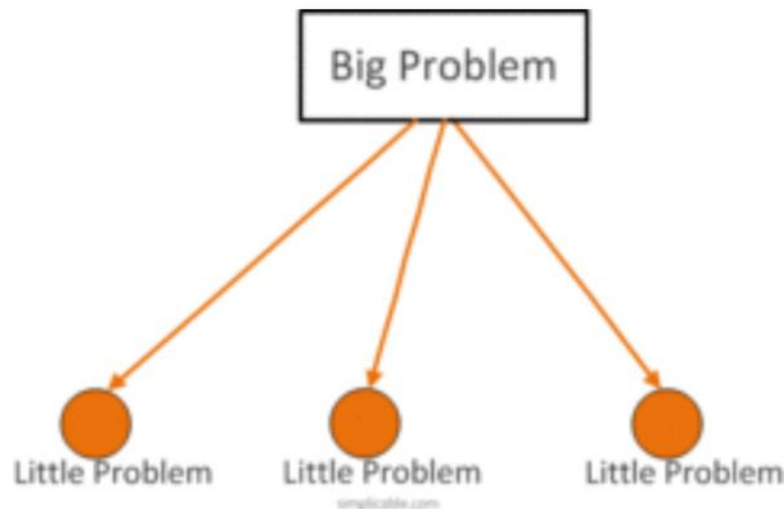
ws-discovery

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 reusable general.

Empatía

Componer diferentes capacidades de negocio.



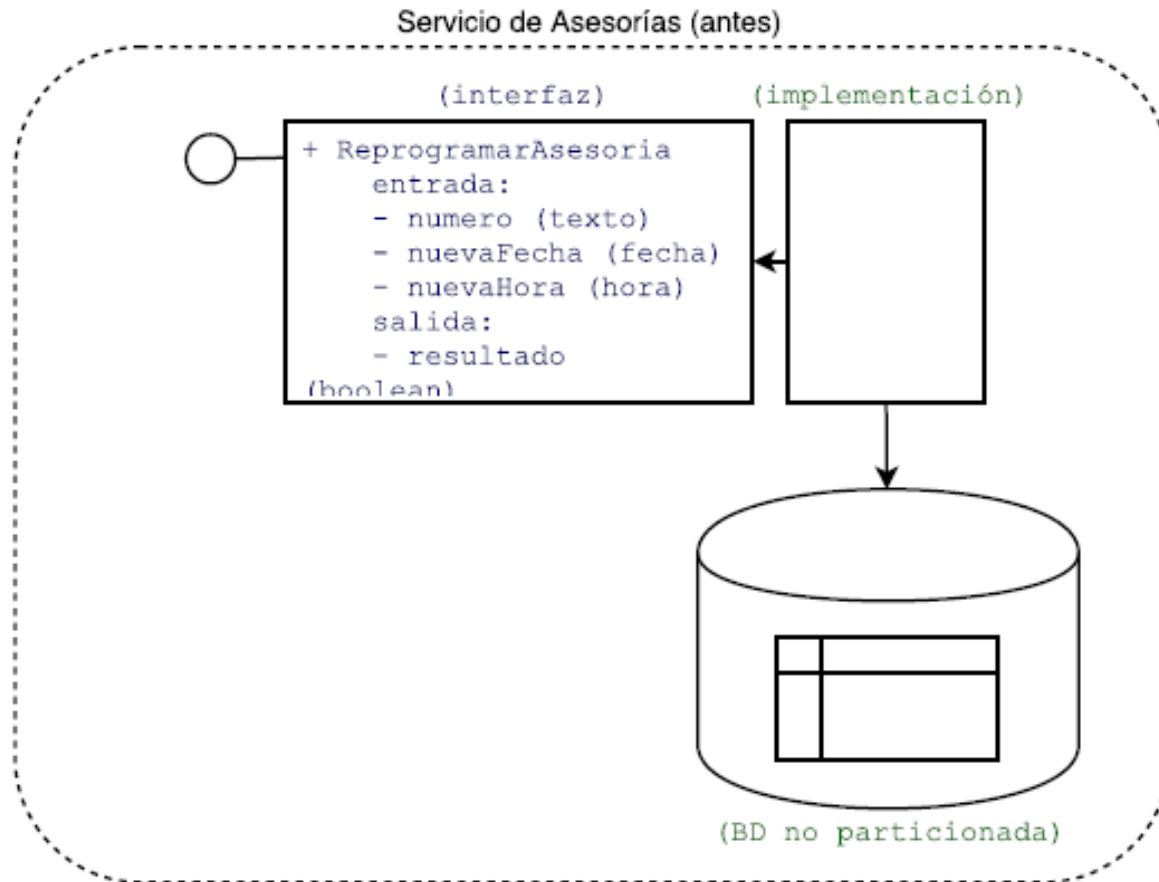
Composición: Automatización de un proceso o tarea de negocio.

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

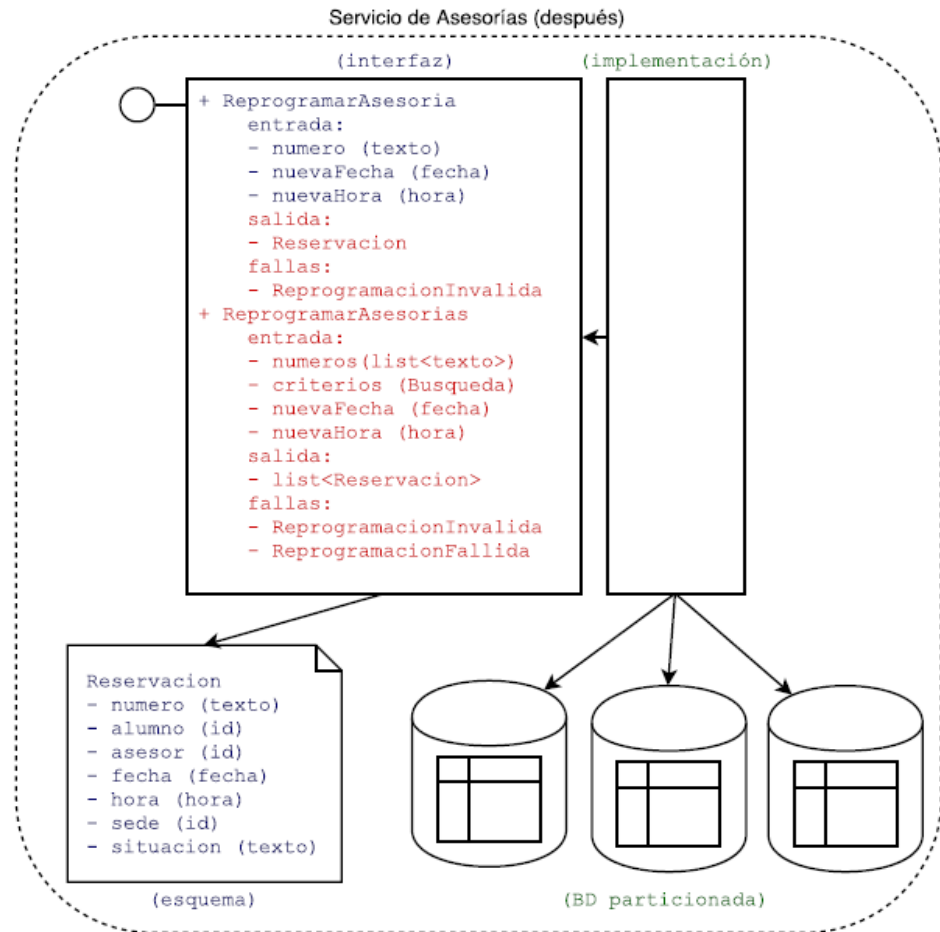
Principio 8: Facilidad de Composición



Caso: Un servicio con poca capacidad de ser compuesto



Solución 8: Un servicio con mayor capacidad de ser compuesto



Resumen

