

Ingeniería de Aplicaciones Web

Diego C. Martínez

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Web Service

¿qué es un *servicio web*?



En términos generales,
un servicio web es una **aplicación accedida remotamente** usando **protocolos de Internet**, y que utiliza **XML(JSON)** como mecanismo de mensajes.

No tiene dependencias de ningún sistema operativo o lenguaje de programación.



Web Service



XML-RPC / JSON-RPC

Invocación remota de métodos

XML (JSON) como lenguaje independiente de las plataformas

Estructura de mensajes predefinida

SOAP

Evolución de XML-RPC

Messaging Framework

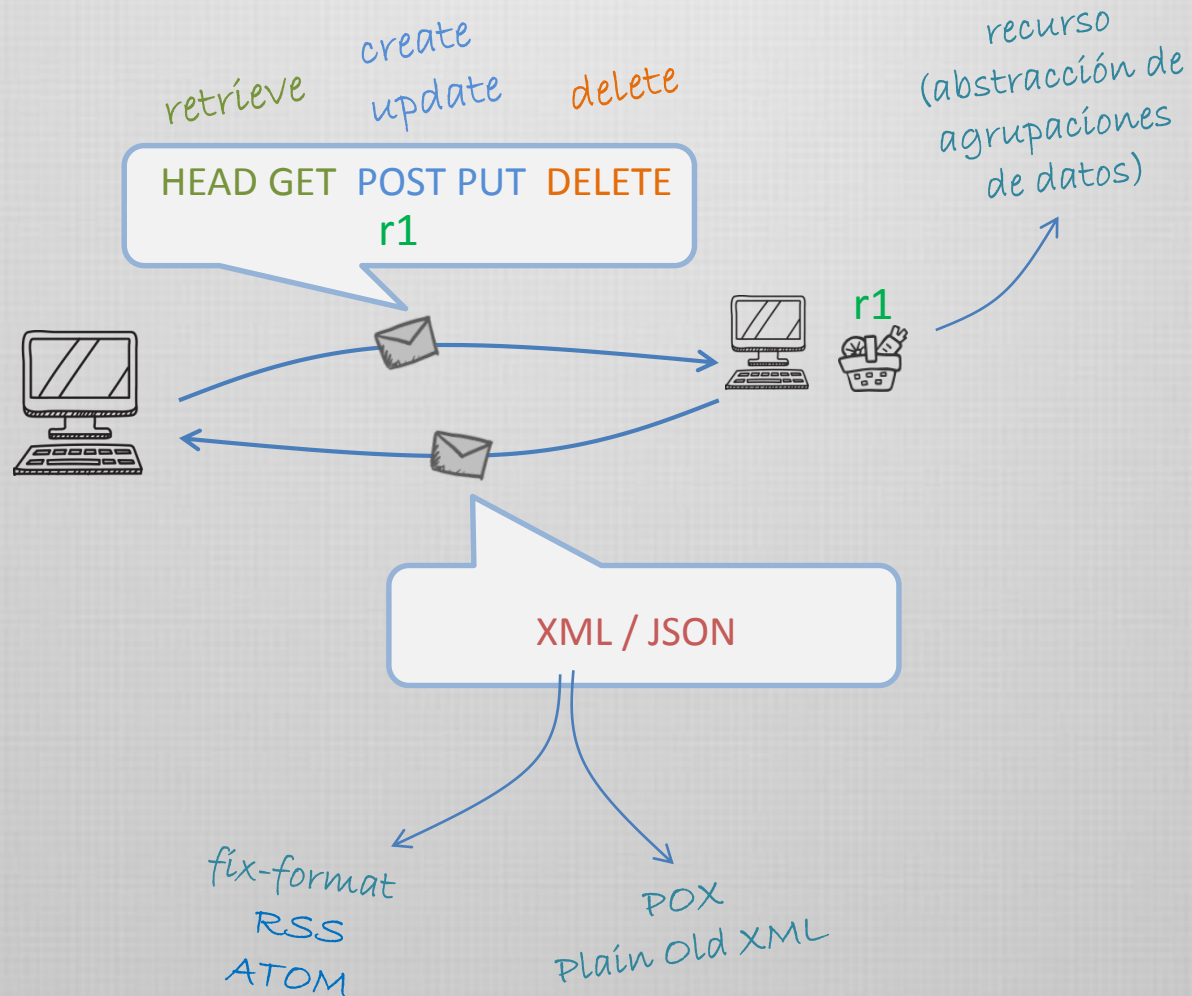
Estructura de mensajes predefinida

REST

Reinterpretación de la relación entre verbos HTTP y URIs

No hay estructura predefinida del contenido de los mensajes

REST



Verbos HTTP

Los verbos HTTP son las **acciones** del cliente REST

Uniform Interface

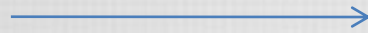
→ **GET, POST, PUT, DELETE, HEAD**

TIPS {
GET **no debe modificar** el estado del recurso.
Preferir **POST** para la creación de recursos.
Usar **PUT** cuando el cliente decide el URI
DELETE debe ser idempotente (como demanda HTTP spec)

Verbo HTTP	/clientes	/clientes/{id}
GET	200 OK, lista de clientes.	200 OK, un cliente. 404 Not Found, id inválido o no encontrado
PUT	404 Not found, a menos que se desee cambiar la colección	200 OK o 204 No content. 404 Not Found, id inválido o no encontrado
POST	201 Created. Header Location con el URI del nuevo recurso	404 Not found
DELETE	404 Not Found, a menos que se desee eliminar la colección	200 OK, 404 Not Found, id inválido o no encontrado.

Nombres de recursos

RESTFul API



colección de URIs

la elección de nombres es importante



sustantivos

"a thing"

nombres predecibles, jerárquicos

POST http://www.example.com/customers/33245/orders/8769/lineitems

GET|PUT|DELETE http://www.example.com/products/66432

GET http://api.example.com/services?op=update_customer&id=12345&format=json

GET http://api.example.com/customers/12345/update

<https://developers.facebook.com/docs/graph-api>

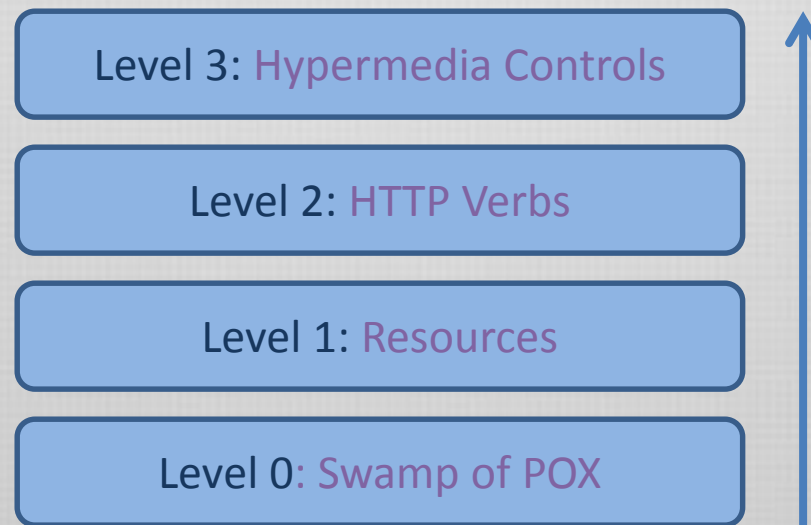
REST - URLs

Algunas guías generales para la estructuración de las URLs

- Pensar en la URL como una interfaz auto-documentada.
Mantener una estructura jerárquica razonable y descriptiva
- Ocultar la tecnología de scripting del lado servidor (`.jsp`, `.asp`, `.php`, etc)
Utilizar si es posible, URL-rewriting.
- Utilizar siempre letras minúsculas
Es una convención general y evita confusiones.
- Sustituir espacios por guiones
Hace legible el URL y evita encodings que serían obligatorios
- Evitar, en lo posible, los query strings.
La excesiva parametrización ofusca el URL.
Puede utilizarse también URL-rewriting
- En lo posible reemplazar el mensaje 404 por un recurso por default
Es más general y conduce a un procesamiento uniforme del lado cliente

Richardson Maturity Model

Es una forma de calificar la API según las restricciones REST
Cuanto más adhiera el API a las restricciones, mayor calificación



Richardson Maturity Model

Level 3: **Hypermedia Controls**

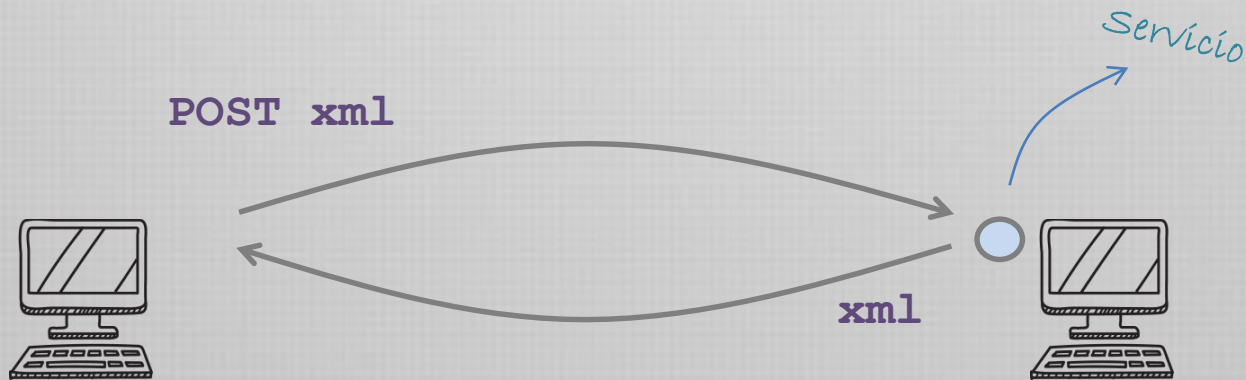
Level 2: **HTTP Verbs**

Level 1: **Resources**

Level 0: **Swamp of POX**

Básicamente, usar HTTP para intercambiar *Plain Old Simple XML*

XML-RPC está en este nivel



Richardson Maturity Model

Level 3: **Hypermedia Controls**

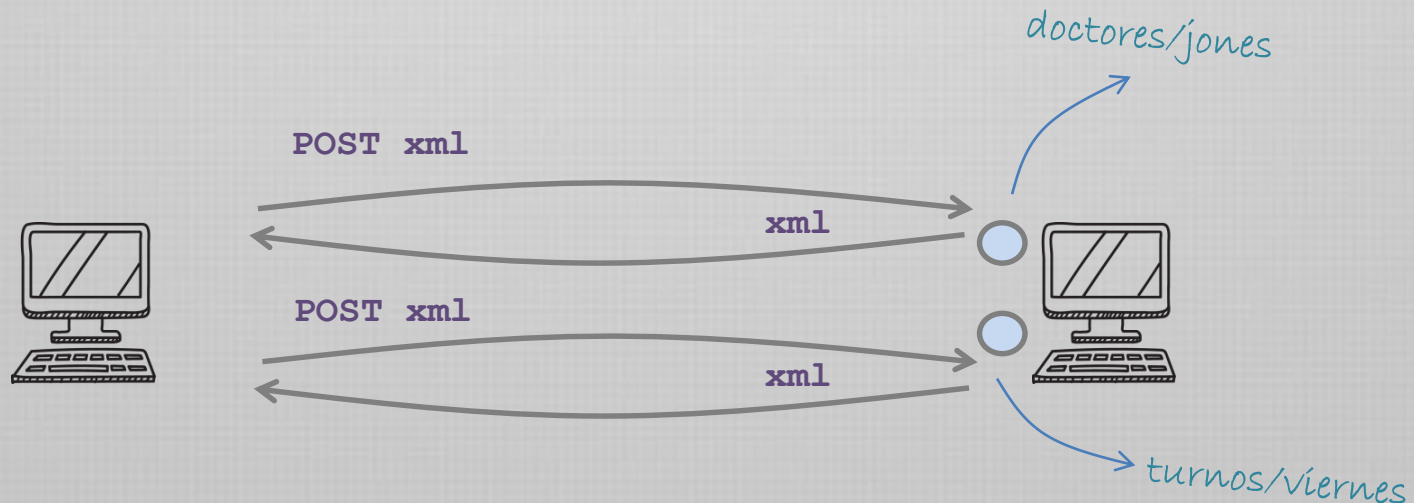
Level 2: **HTTP Verbs**

Level 1: **Resources**

Level 0: **Swamp of POX**

En lugar de realizar pedidos a un endpoint particular de un servicio, interactuamos con recursos individuales

Se identifican los recursos y se separan de acuerdo a su utilidad



Richardson Maturity Model

Level 3: **Hypermedia Controls**

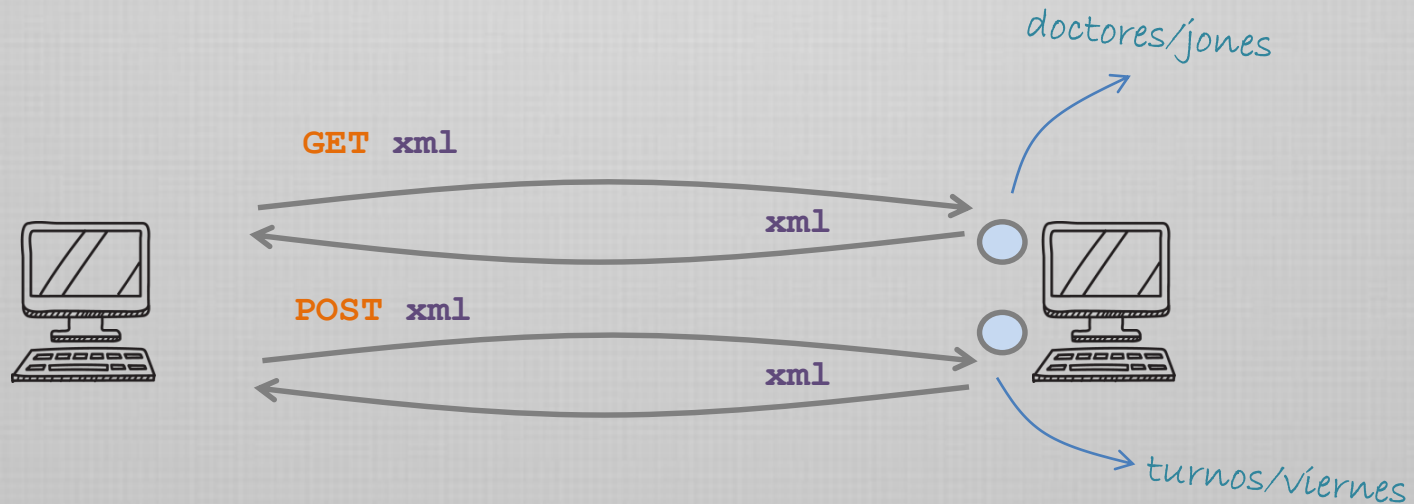
Level 2: **HTTP Verbs**

Level 1: **Resources**

Level 0: **Swamp of POX**

Utilizar los verbos HTTP apropiadamente, de manera similar a la mecánica web en general

Similar a CRUD



Richardson Maturity Model

Level 3: Hypermedia Controls

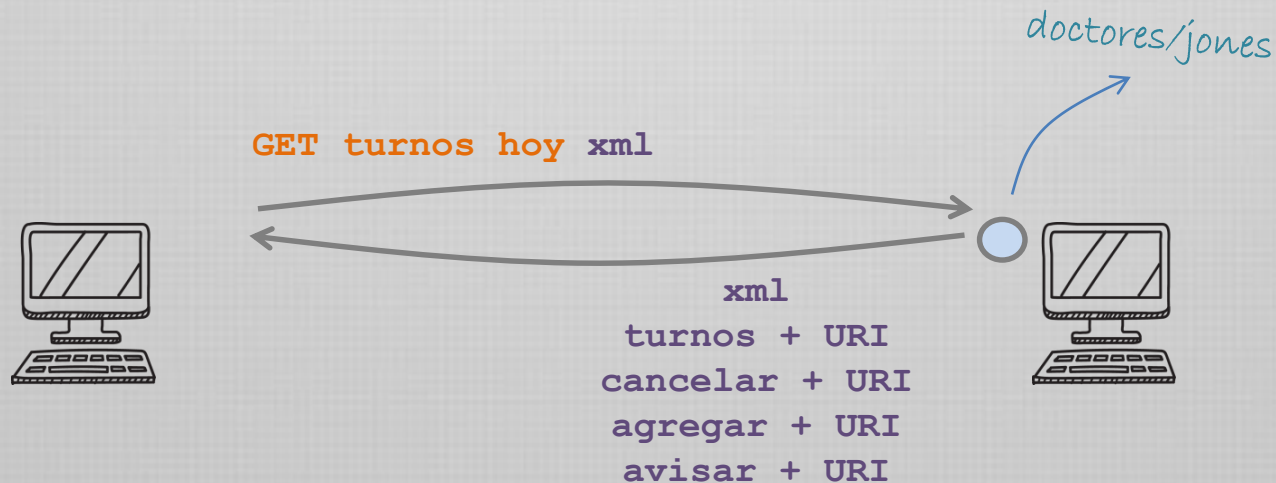
Level 2: HTTP Verbs

Level 1: Resources

Level 0: Swamp of POX

Hypermedia Controls indica que los datos deben contener información sobre las posibles siguientes acciones

Incluye el URI de los recursos próximos a solicita



JAVA WS

JAX-RS

Java API for RESTful web services

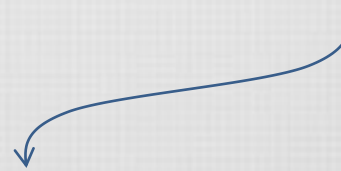
Provee facilidades para la implementación de servicios web RESTful

“The Java API for RESTful Web Services provides portable APIs for developing, exposing and accessing Web applications designed and implemented in compliance with principles of REST architectural style.”

javax.ws.rs
javax.ws.rs.client
javax.ws.rs.container
javax.ws.rs.core
javax.ws.rs.ext

JAVA WS

JAX-RS hace uso intensivo de anotaciones



meta-datos agregados al código
representan usualmente “boilerplate code”

*código repetitivo,
generalmente sin
alteraciones importantes*

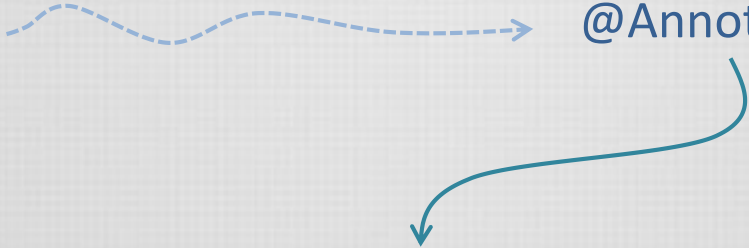
La máquina virtual JVM identifica
las anotaciones, deduce el
comportamiento deseado y genera
el código necesario

Las anotaciones son aplicables a
declaraciones de paquetes,
declaraciones de tipos, constructores, métodos ,
atributos, parámetros y variables.

Anotaciones

Anotaciones en Java

@AnnotationID



Información para el compilador:
usadas por el compilador para detectar errores o
suprimir warnings.

Compile-time and deployment-time processing:
Las herramientas de desarrollo procesan las anotaciones
para generar código, archivos XML, etc.

Runtime processing:
algunas anotaciones son examinadas en
tiempo de ejecución.

Existen anotaciones predefinidas
`@Deprecated`, `@Override`, `@SuppressWarnings`

Es possible crear nuevas anotaciones....

Anotaciones

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {
    public boolean enabled() default true;
}
```

```
public class TestExample {
    @Test
    void testA() {
        ...
    }

    @Test(enabled = false)
    void testB() {
        ...
    }
}
```

Anotaciones

```
Class<TestExample> obj = TestExample.class;
for (Method method : obj.getDeclaredMethods()) {

    if (method.isAnnotationPresent(Test.class)) {
        Annotation annotation = method.getAnnotation(Test.class);
        Test test = (Test) annotation;
        if (test.enabled()) {
            try {
                method.invoke(obj.newInstance());
                System.out.printf("OK");
                passed++;
            } catch (Throwable ex) {
                System.out.printf("FAILED"+method.getName());
                failed++;
            }
        }
    }
}
```

Anotaciones JAX-RS

@Path

especifica el path relativo a la clase o método.

@GET, @PUT, @POST, @DELETE, @HEAD

especifica el tipo de request HTTP para el recurso.

@Produces

especifica la respuesta en Internet media types

@Consumes

especifica el tipo del dato aceptado en el request, como Internet media types.

```
@GET
```

```
@Path("saludo")
```

```
@Produces("text/html")
```

```
public String getHtml() {
```

```
    return "<html><body><h1>Hola!!</body></h1></html>";
```

```
}
```

Anotaciones JAX-RS

```
@Path("/libros")
public class Biblioteca {

    @GET
    @Path("/bestsellers")
    @Produces("text/xml")
    public String show() {
        //mostrar libros
    }

    @POST
    @Path("/bestsellers")
    @Consumes("text/xml")
    public String save(String s;) {
        //guardar libro
    }

}
```

Arquitecturas Orientadas a Servicios

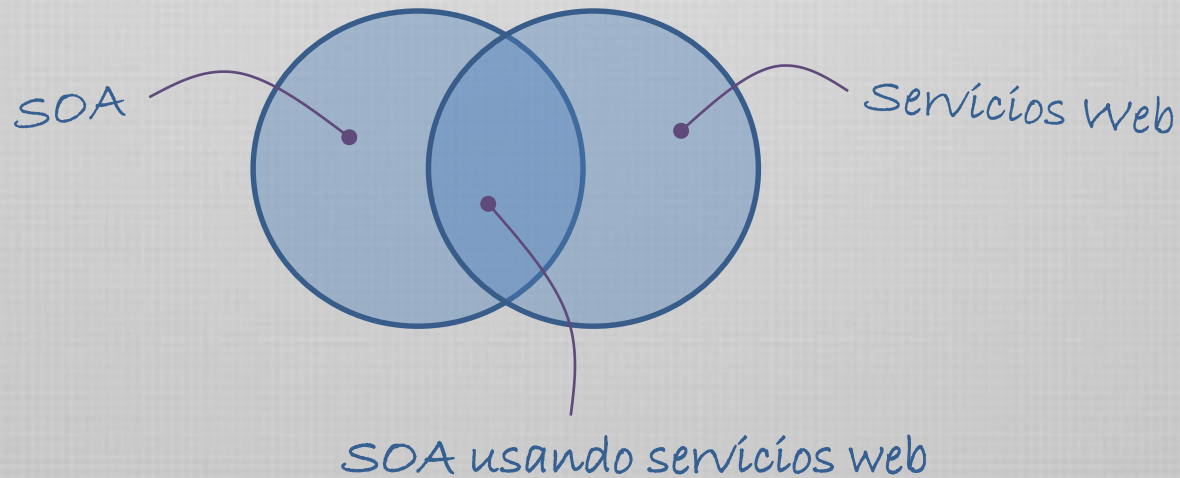
SOA

Service Oriented Architecture



una forma de diseñar, implementar y ensamblar servicios
(para sustentar "business functions")

primeras SOAs en los 90's (DCOM, CORBA)

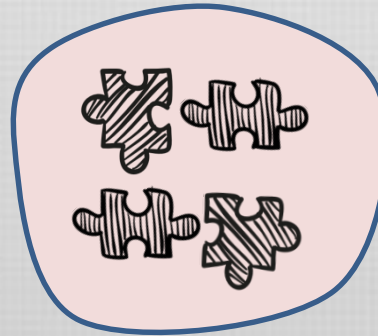


Arquitecturas Orientadas a Servicios

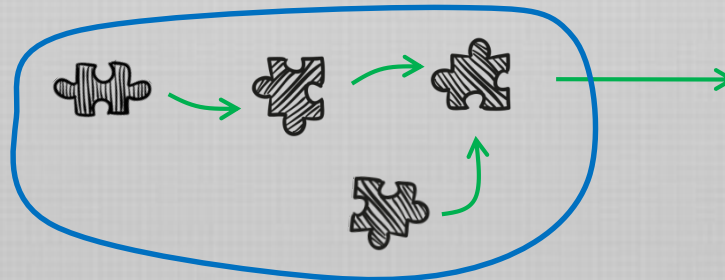
SOA

Service Oriented Architecture

El software es particionado en función de **servicios**
poco acoplados
que ocultan su implementación



El software ofrece **funcionalidad** ensamblando servicios



Arquitecturas Orientadas a Servicios

SOA

Service Oriented Architecture



servicios

**servicios
atómicos**

función bien definida
y auto-contenida que
no depende de otros
servicios

**servicios
compuestos**

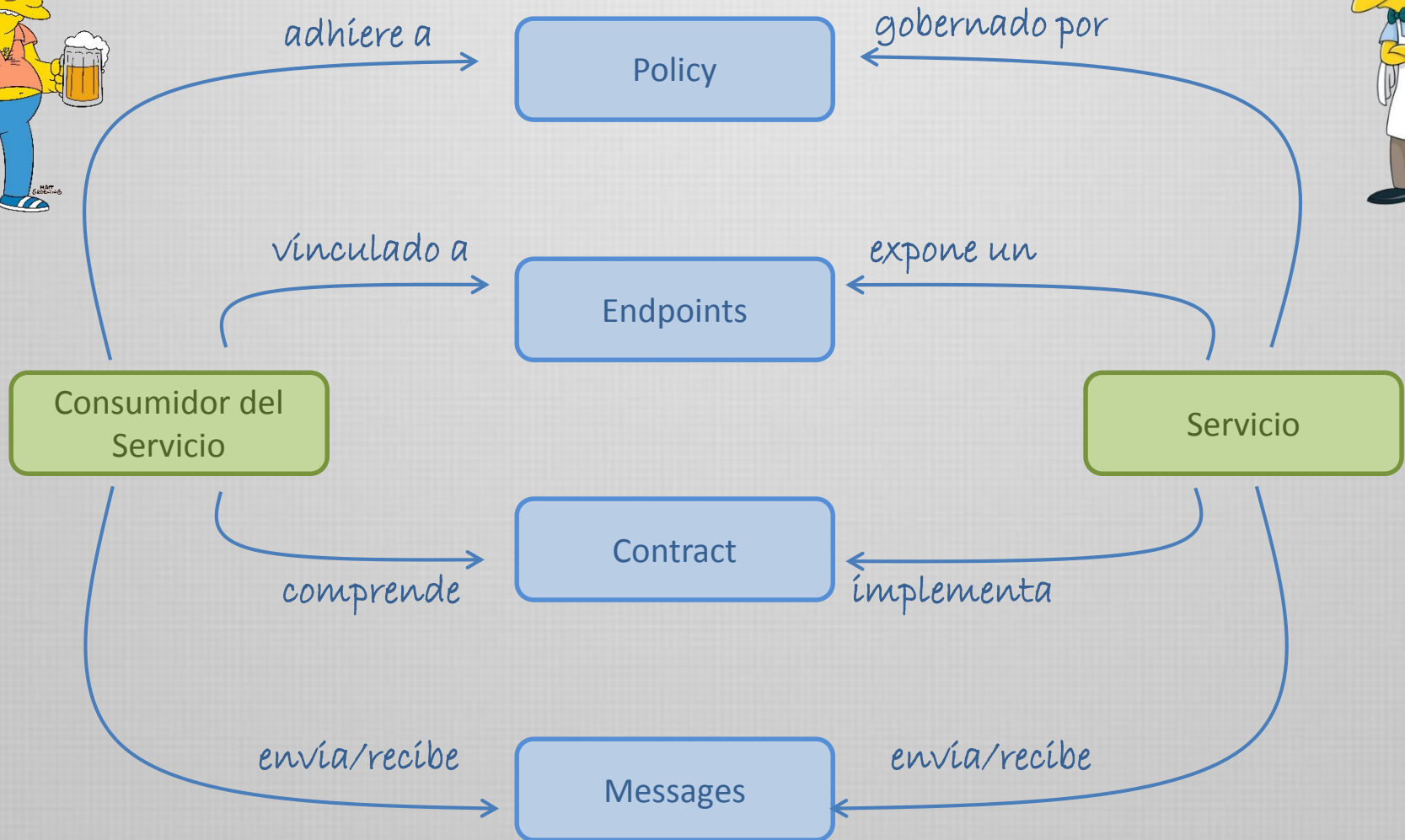
"compound services"
ensamblado de
servicios atómicos o
compuestos

**servicios
poco acoplados**

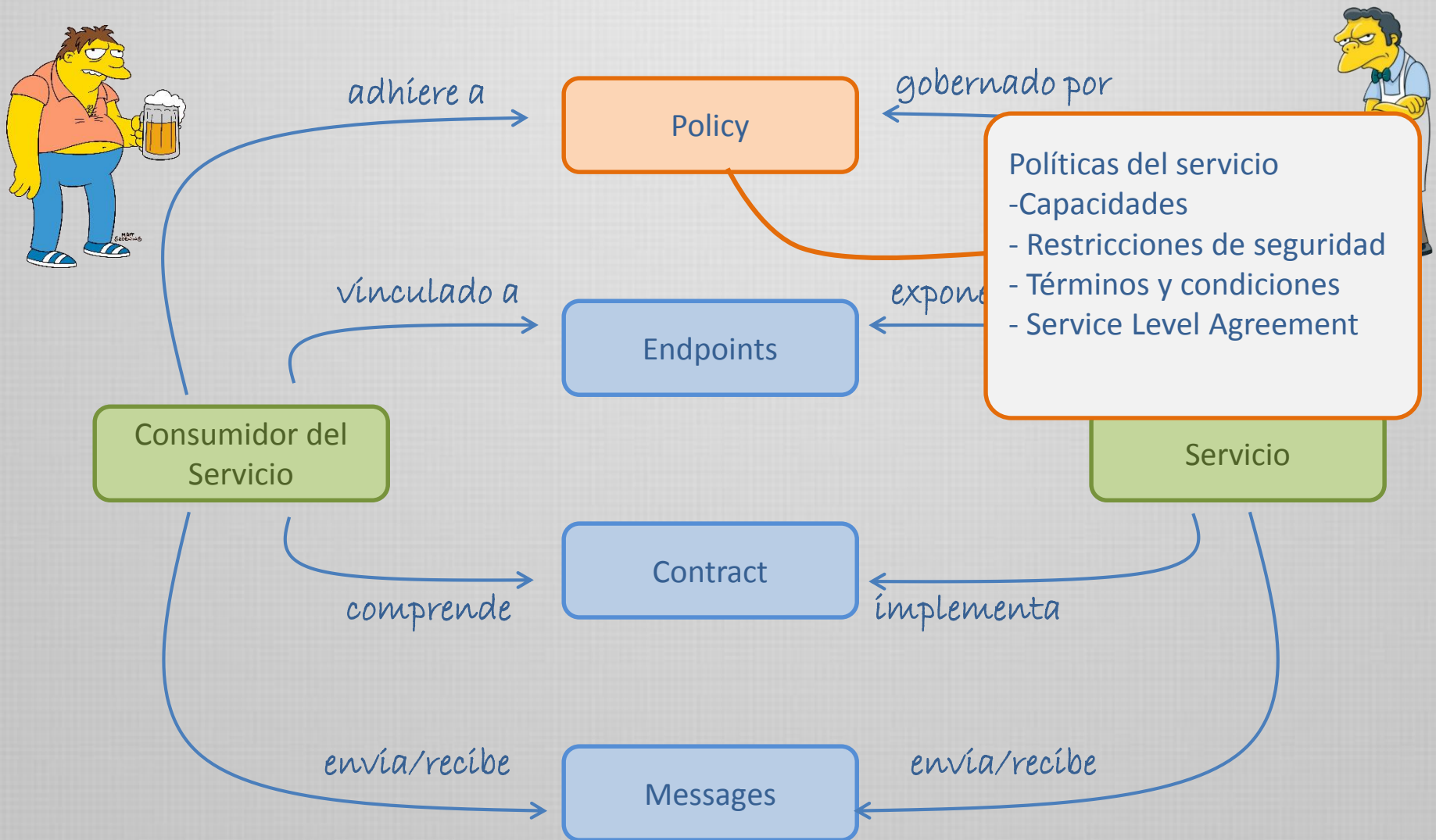
servicios con
abstracción de
implementación

El diseño de los servicios requiere un balance entre atómicos-compuestos, manteniendo los servicios con poco acoplamiento

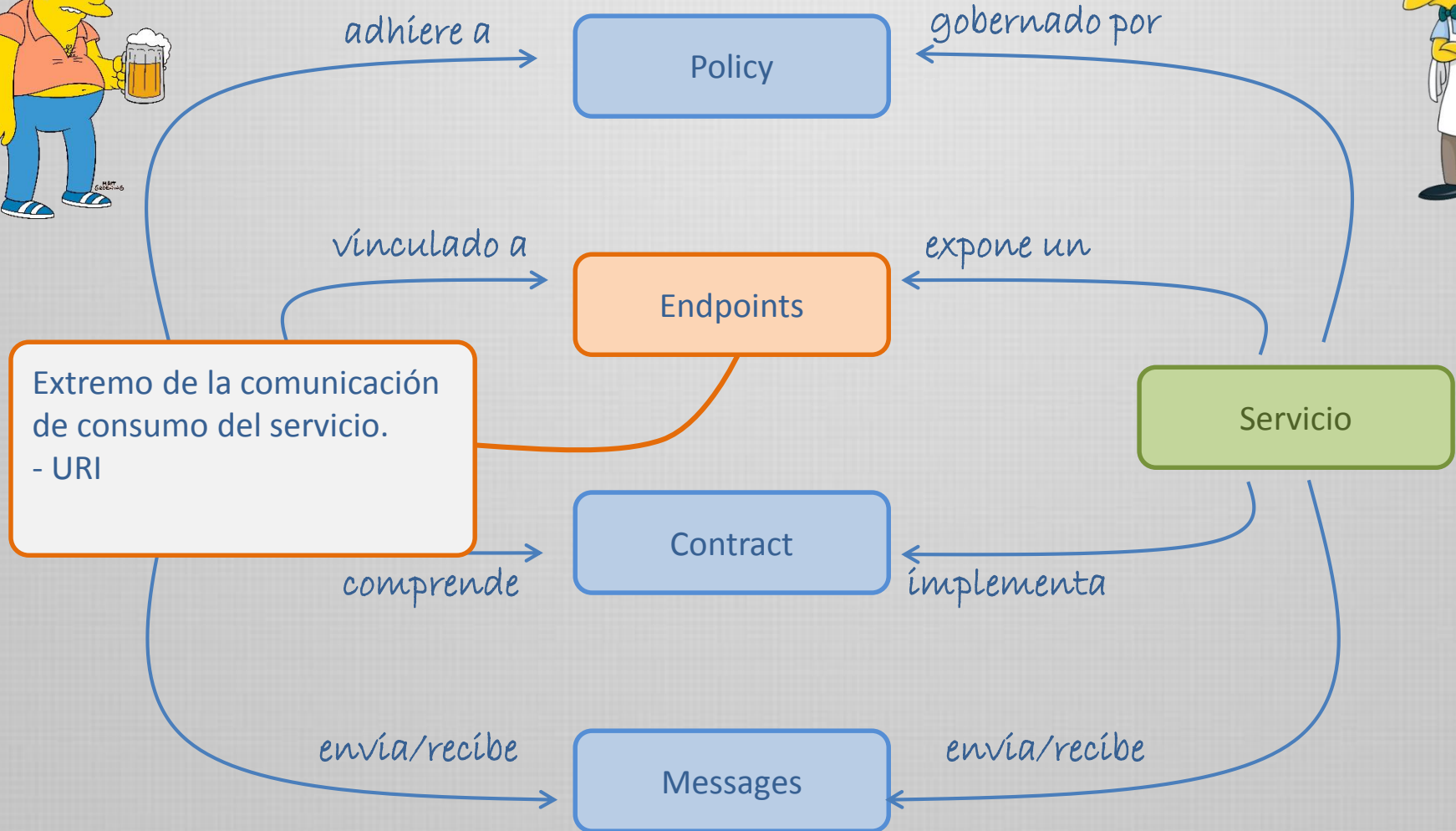
SOA - Elementos



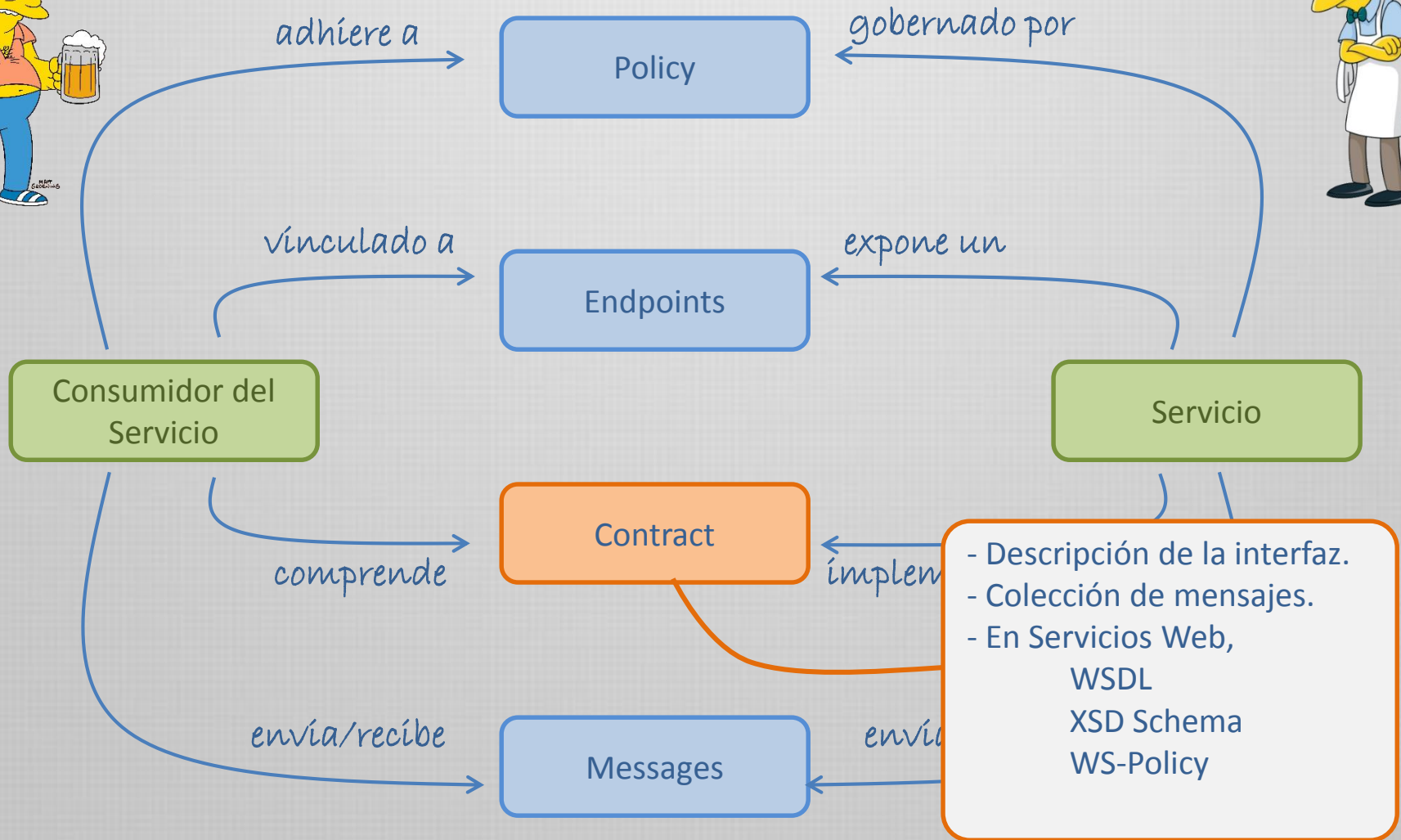
SOA - Elementos



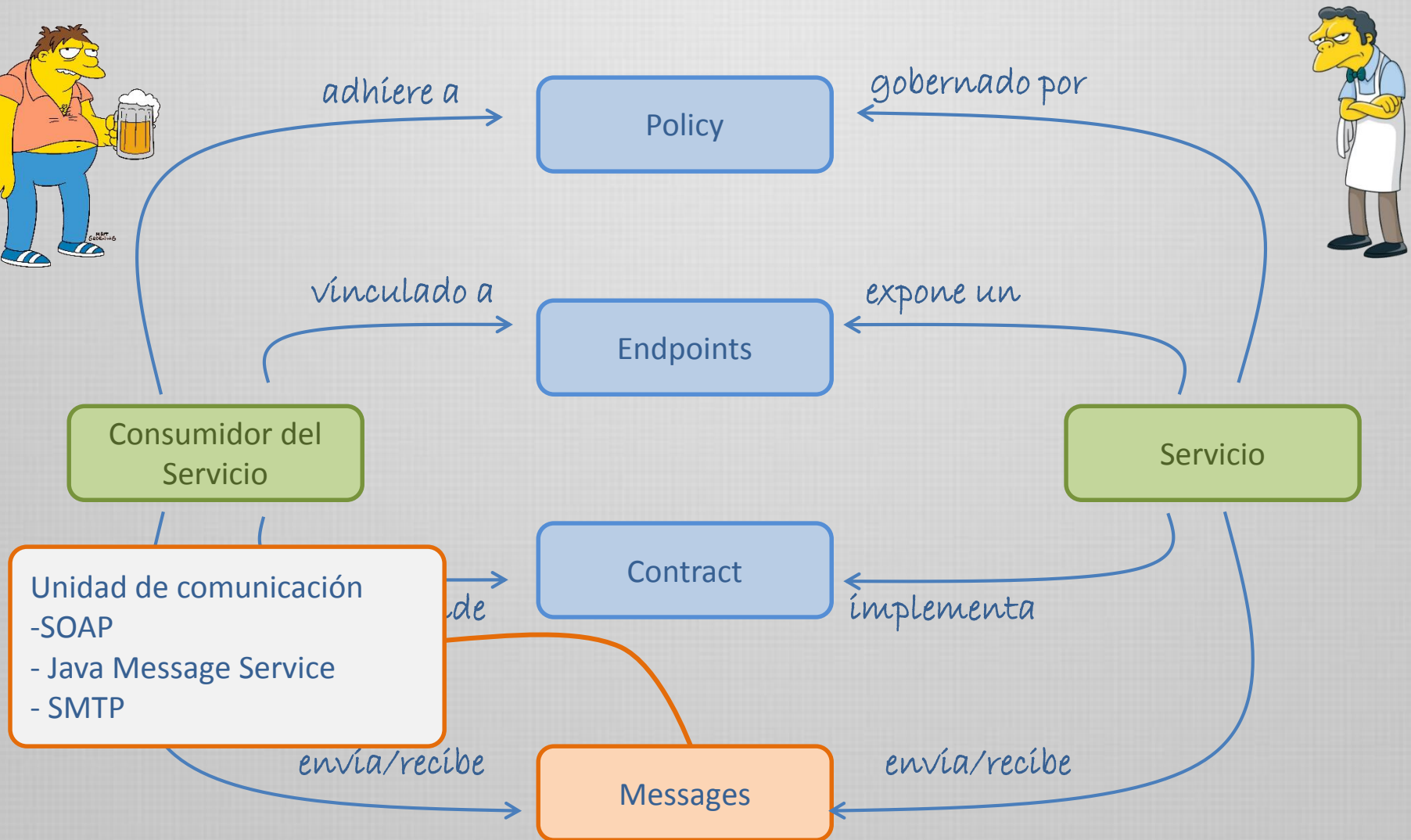
SOA - Elementos



SOA - Elementos



SOA - Elementos



SOA - Beneficios

Flexibilidad Organizacional

Es fácil integrar componentes con otras aplicaciones

Interoperabilidad

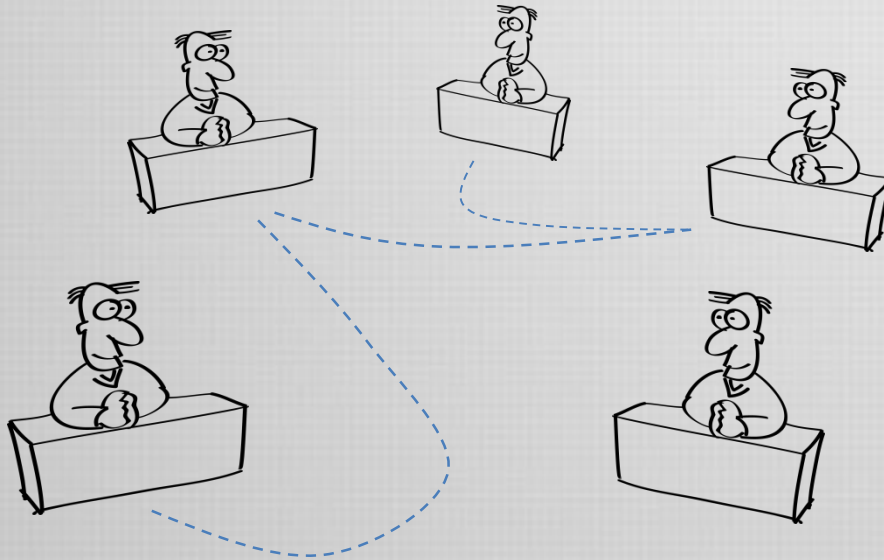
Las partes se comunican entre sí independientemente de su naturaleza tecnológica

Reusabilidad

Los servicios pueden ser consumidos por otros servicios

Mantenimiento

Cambios internos en un servicio no afectan el resto del sistema.



Arquitecturas Orientadas a Servicios

Consideraciones de Diseño

Adoptar estándares

servicios web, vocabularios semánticos, estructuras de datos

Encapsular aplicaciones *legacy* con interfaces

ofrecer servicios web para accederlas

Ocultar la estructura de los datos subyacentes

provee flexibilidad y desacoplamiento de vendors.

Diseñar servicios para la reusabilidad

establecer apropiadamente la granularidad de los servicios

Rastrear el uso de los servicios ofrecidos

tokens, versioning . Amazon Web Services como ejemplo

Mensajes pesados dentro del servicio, mensajes livianos entre servicios

la comunicación es una sobrecarga a tener en cuenta

Principios de Arquitecturas de Servicios



Don Box

“Four Tenets in Service Architecture”

Fronteras explícitas

Los servicios interactúan por mensajes a través de fronteras explícitas que protegen/ocultan las implementaciones internas

Los servicios son autónomos

El servicio reacciona ante un mensaje. No depende del contexto de otros servicios. Son independientes del sistema subyacente.

Los servicios comparten schema y contratos, no clases

*Agnosticismo de tecnologías de programación subyacente.
Sólo tecnologías como XML, WSDL*

La compatibilidad de servicios se basa en políticas.

*policy = descripción de capacidades y requerimientos de cada web service.
WSDL no siempre es suficiente.*