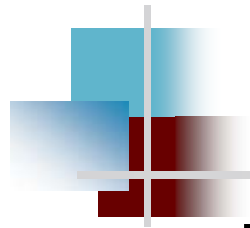




Tema 5.

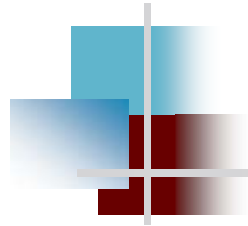
Servicios Web

Miguel A. Laguna



Objetivos

- Explicar el concepto de Servicios Web
- Presentar los estándares propuestos de Servicios Web
- Conocer las posibilidades de las herramientas para la automatización y cumplimiento de los estándares
- Explicar las diferencias entre servicios estándar SOAP y la alternativa RESTful



Desarrollo del tema

- Introducción a los Servicios Web
- Estándares SOAP, WSDL y UDDI
- Servicios RESTful
- REST: Detalles de implementación



Bibliografía

- 📖 Sommerville, I. "Software Engineering" Addison-Wesley, 2010 (9ª ed.)
- 📖 Martin Kalin. Java Web Services: Up and Running. O'Reilly, 2009
- 📖 Leonard Richardson, Sam Ruby. RESTful Web Services, Web services for the real world. O'Reilly, 2007.

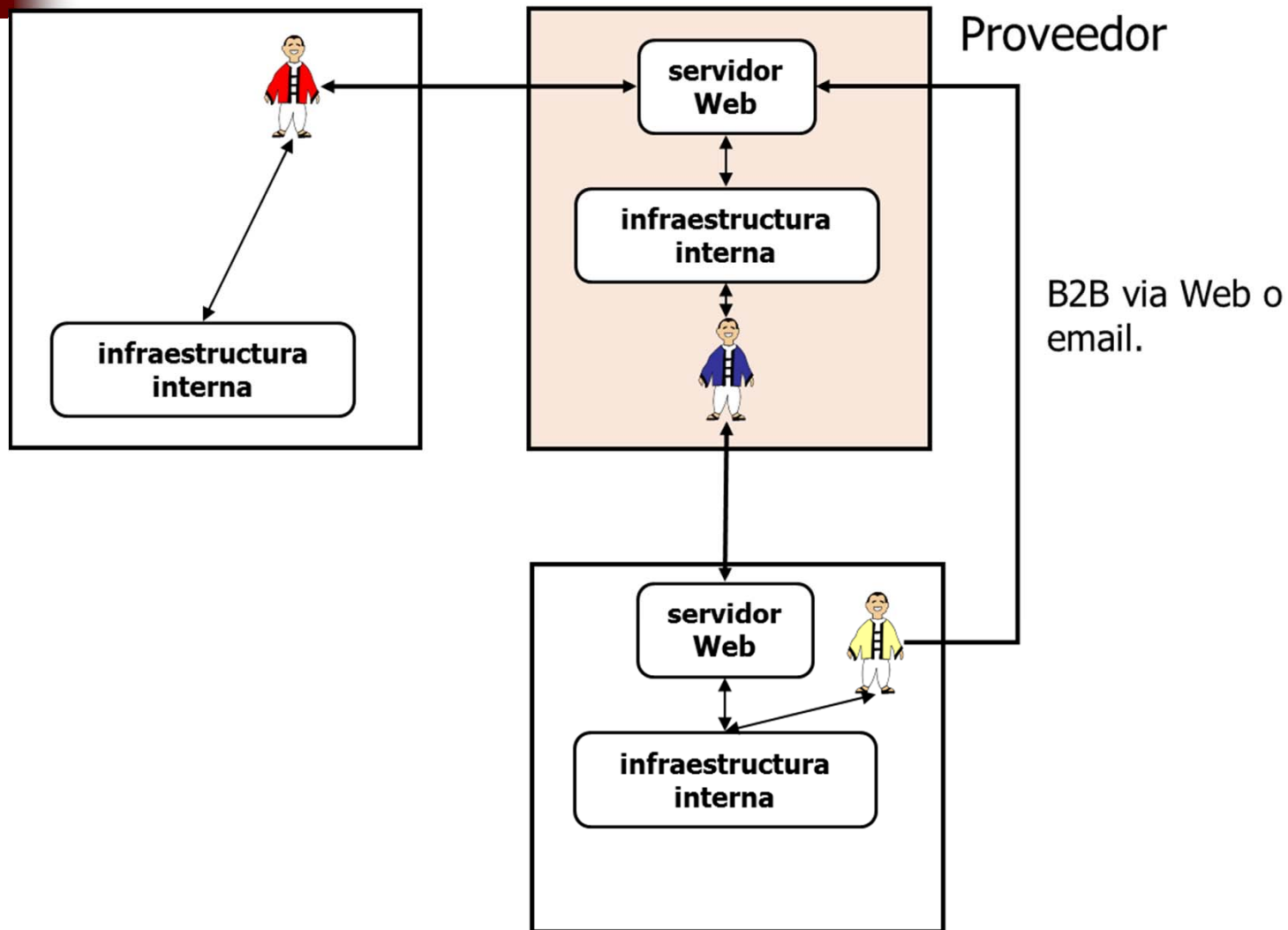
Lecturas complementarias

- 📖 Leonard Richardson and Mike Amundsen, RESTful Web APIs, O'Reilly, 2013

5.1. Introducción a los Servicios Web



El problema: interacción entre organizaciones





Arquitecturas orientadas a Servicios Web

- Un tipo de sistemas distribuidos donde los componentes son servicios autónomos
- Los servicios pueden ejecutarse en diferentes equipos de diferentes proveedores de servicios
- Los sistemas se pueden comunicar de manera directa sin actores humanos (B2B, business to business)

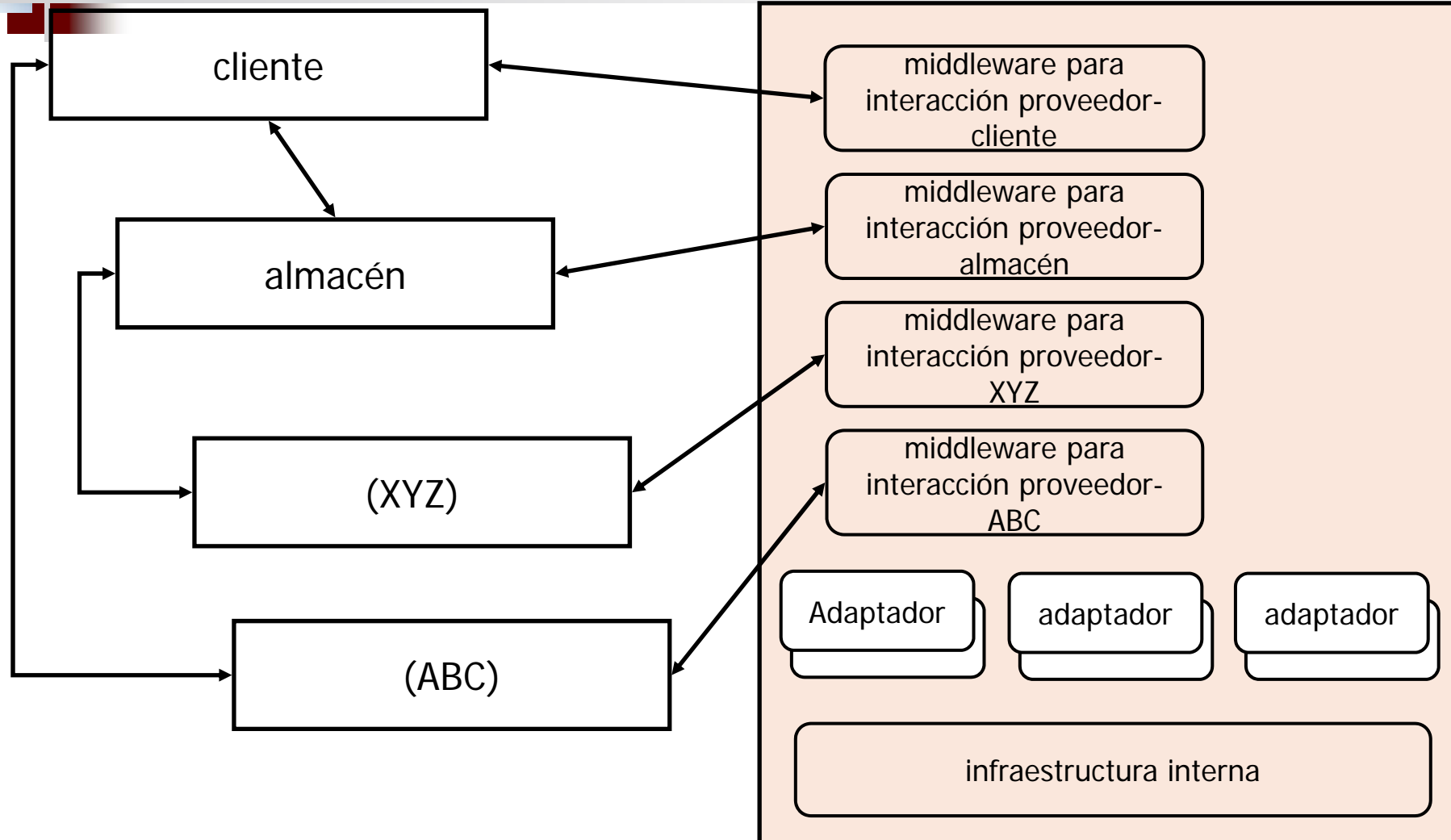


Beneficios de SOA

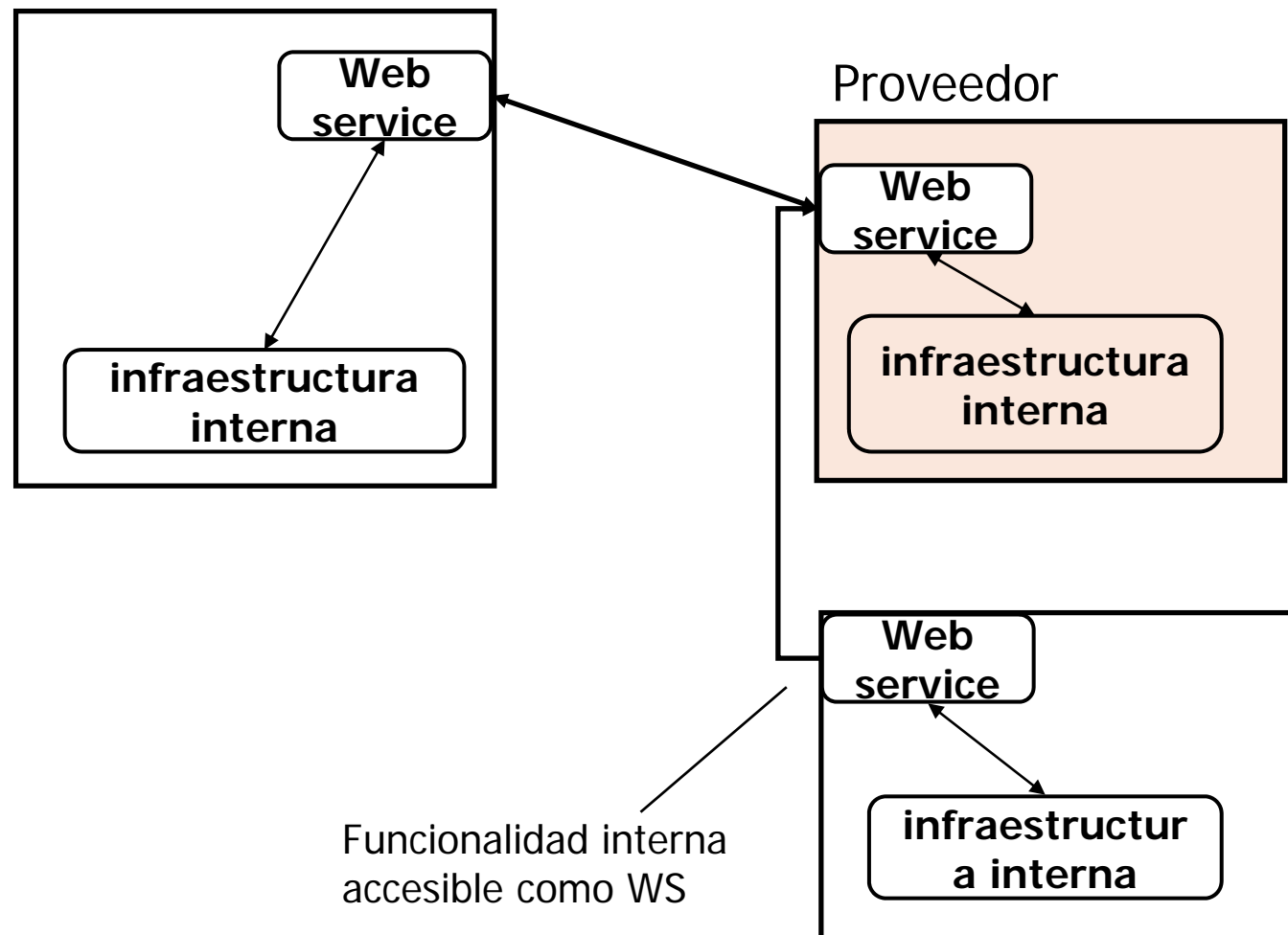
- Los servicios pueden proporcionarse:
 - Localmente
 - Externalizados a proveedores externos.
- Los Servicios son independientes del lenguaje
- La inversión en sistemas legados puede ser preservada
- La computación entre organizaciones se ve facilitada a través del intercambio de información simplificada y estandarizada

Middleware específico

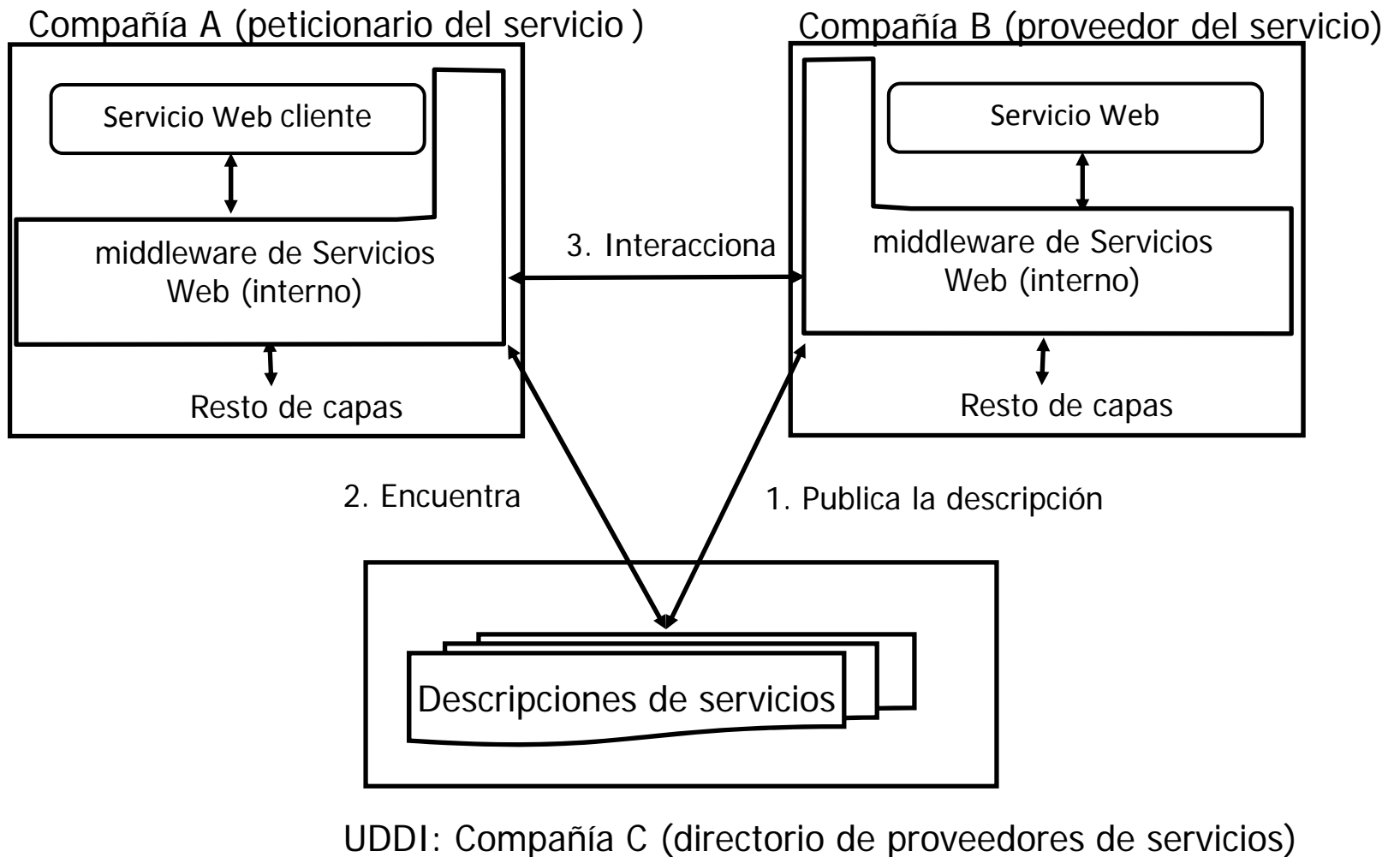
Proveedor



La solución: Servicio Web estándar

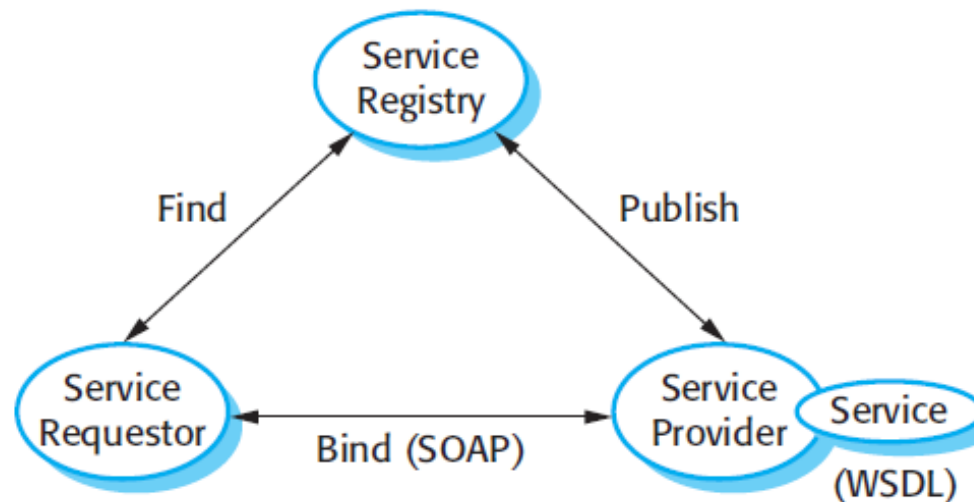


Páginas amarillas...



Arquitecturas orientadas a Servicios Web

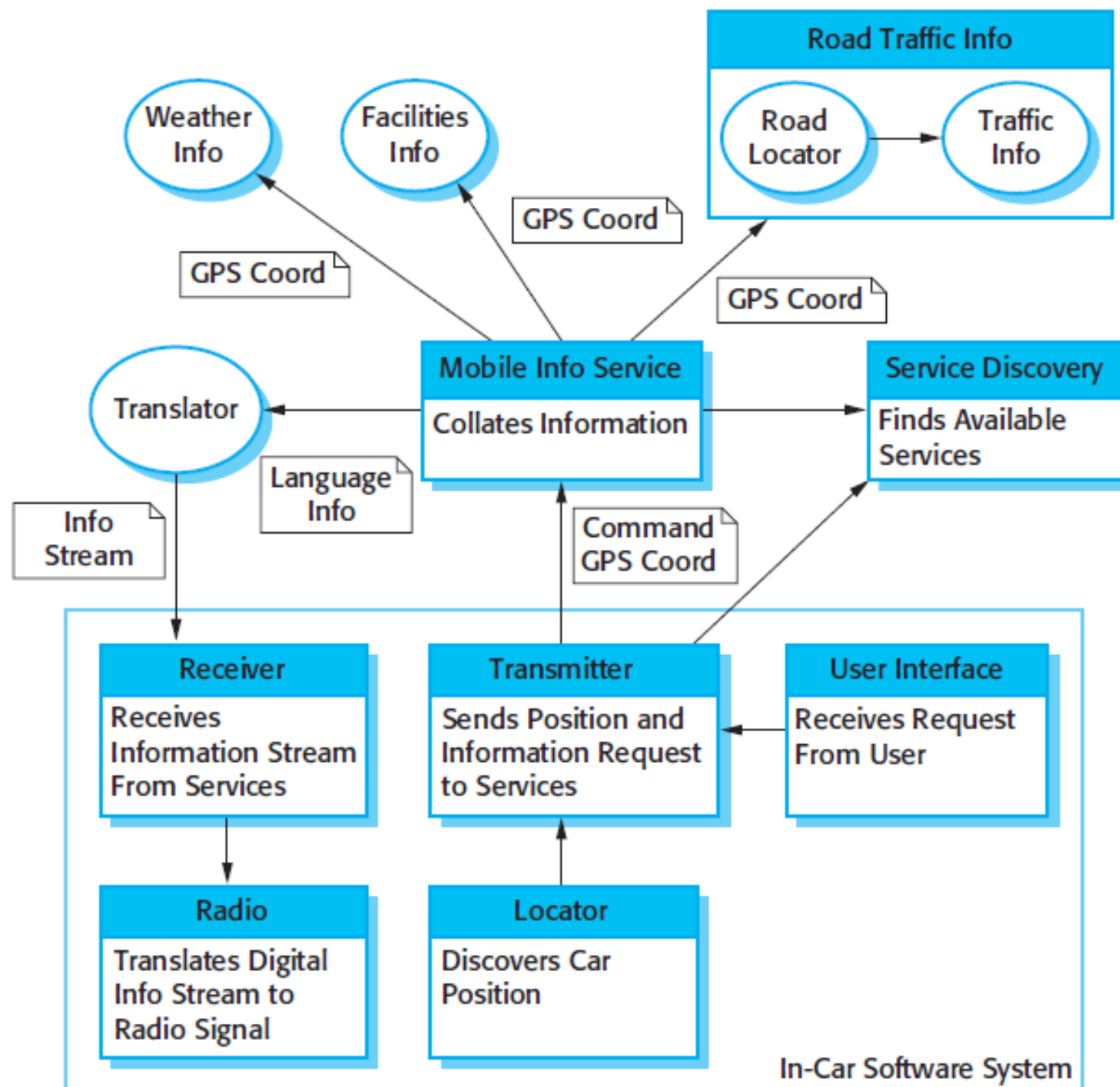
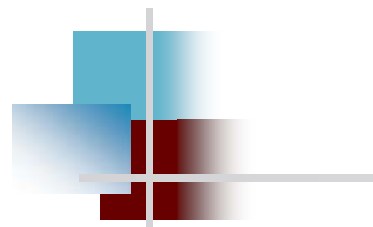
- Protocolos estándar
 - Intercambio de información
 - Definición de servicios
 - Registro de servicios

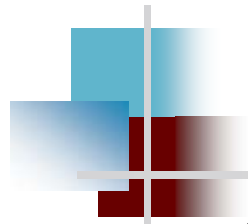




Otro ejemplo de aplicación

- Un sistema de información en el automóvil proporciona a los conductores información sobre el clima, las condiciones del tráfico, información local, etc.
- El sistema está conectado a la radio del coche para que la información se entregue como una señal en un canal de radio específico
- El vehículo está equipado con un receptor de GPS y, con su posición, el sistema accede a una amplia gama de servicios de información.
- La información puede ser entregada en el idioma especificado por el controlador





Ventajas de SOA

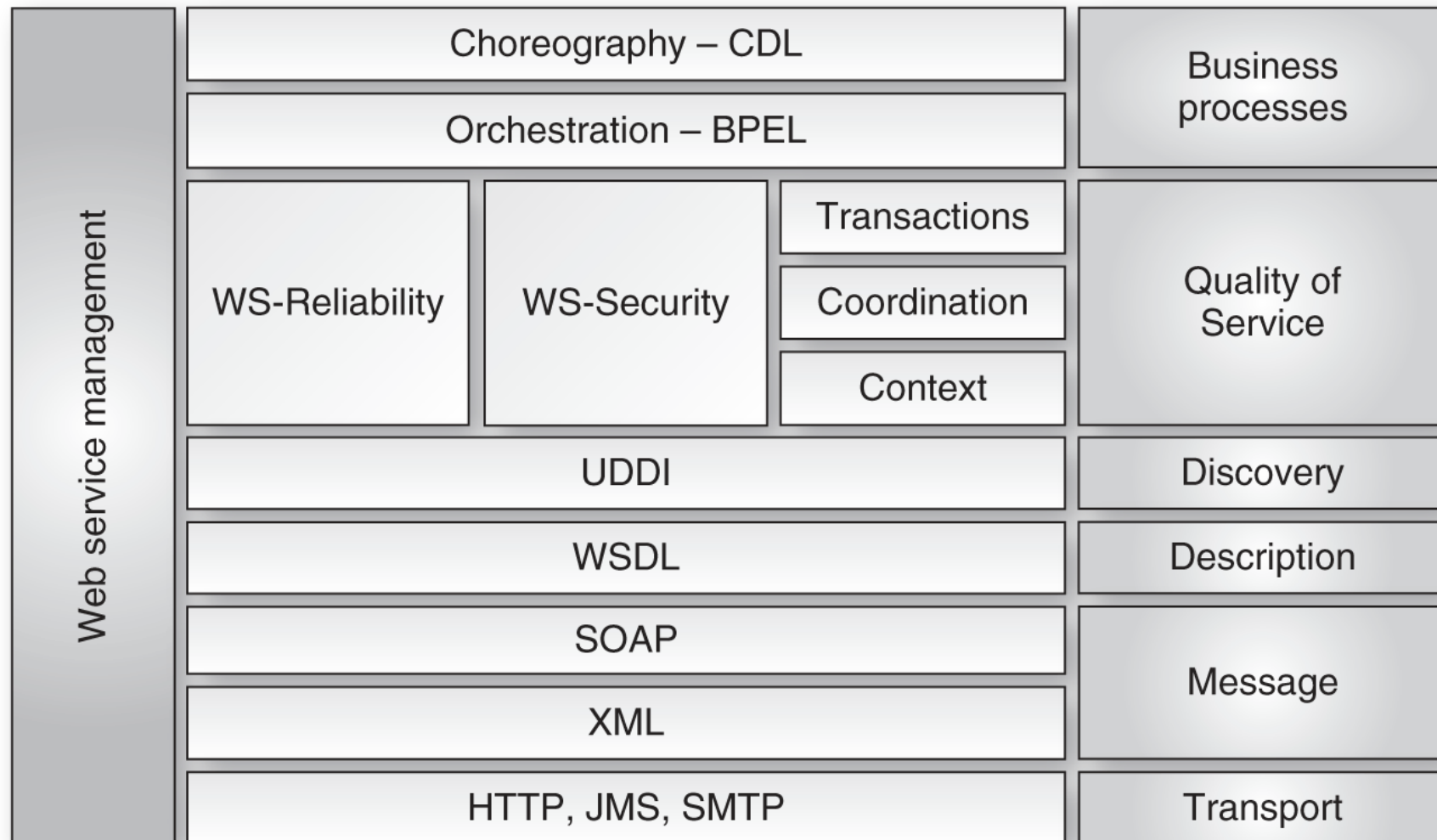
- Cuando el sistema está siendo programado o desplegado no es necesario decidir qué proveedor de servicios se debe utilizar o qué servicios específicos debe utilizar
- Cuando el coche se mueve, el software del sistema In-Car utiliza el servicio de descubrimiento de servicios para encontrar el servicio de información más próximo y acceder a:
 - Información del tráfico
 - Información del tiempo
 - Información de gasolineras, áreas de servicio
- Gracias a la utilización de un servicio de traducción, el sistema puede utilizarse en distintos países y hacer que la información local esté disponible para las personas que no hablan el idioma local

5.2. Estándares de Servicios Web





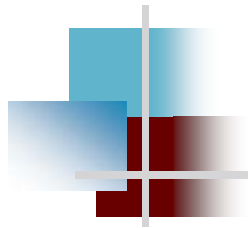
Estándares



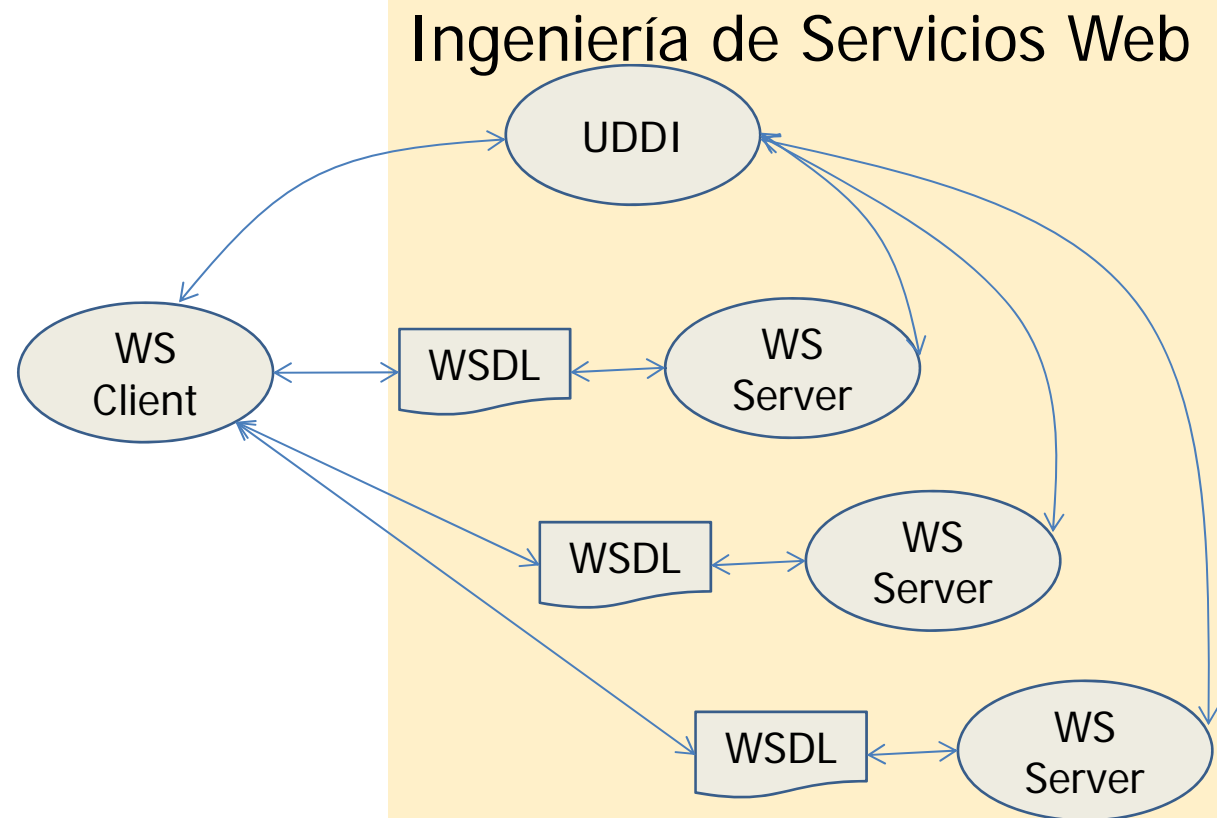


Estándares

- **SOAP** (Simple Object Access Protocol)
 - Estándar de intercambio de mensajes que soporta servicios de comunicaciones
- **WSDL** (Web Service Definition Language)
 - Lenguaje de descripción de interfaces de servicios
- **UDDI** (Universal Description Discovery and Integration)
 - Define la especificación para descubrir la existencia de un servicio
- **WS-BPEL** (Business Process Execution Language)
 - Un lenguaje estándar para la composición de Servicios Web
- **WSCI/CDL** (Web Services Choreography Interface/Choreography DL)
 - Coordinación (incluso entre distintas organizaciones)

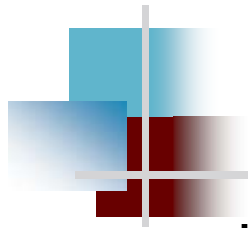


Estándares e Ingeniería



SOA: Ingeniería de Software Orientada a Servicios BPEL

WSGI/CDL



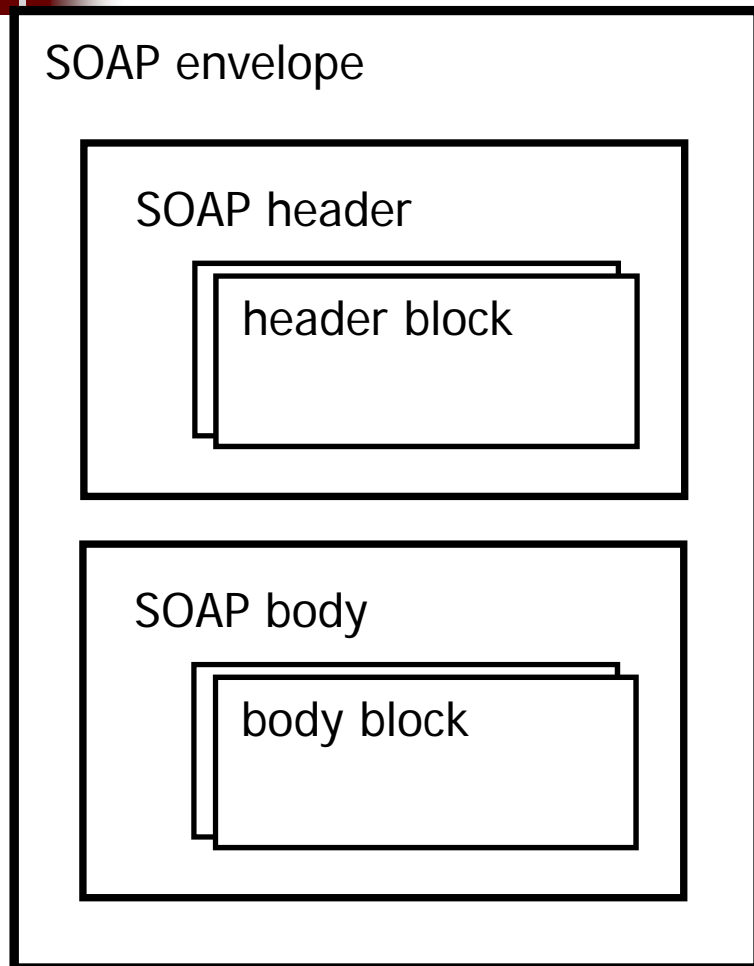
XML es la base

- Un menú como mensaje XML

```
<starter>
  <dish name = "soup" type = "tomato" />
  <dish name = "soup" type = "fish" />
  <dish name = "pigeon salad" />
</starter>
<main course>
  <dish name = "steak" type = "sirloin" cooking = "medium" />
  <dish name = "steak" type = "fillet" cooking = "rare" />
  <dish name = "sea bass">
</main>
<accompaniment>
  <dish name = "french fries" portions = "2" />
  <dish name = "salad" portions = "1" />
</accompaniment>
```



SOAP



- La información (el cuerpo) del mensaje se completa con una cabecera se inserta dentro de un "sobre"



SOAP

```
<?xml version='1.0' ?>
```

```
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope" >
```

```
<env:Header>
```

```
<t:transactionID
```

```
  xmlns:t="http://intermediary.example.com/procurement"
```

```
  env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
```

```
  env:mustUnderstand="true" >
```

```
    57539
```

```
</t:transactionID>
```

```
</env:Header>
```

```
<env:Body>
```

```
<m:orderGoods
```

```
  env:encodingStyle="http://www.w3.org/2002/06/soap-encoding"
```

```
  xmlns:m="http://example.com/procurement">
```

```
<m:productItem>
```

```
  <name>ACME Softener</name>
```

```
</m:productItem>
```

```
<m:quantity>
```

```
  35
```

```
</m:quantity>
```

```
</m:orderGoods>
```

```
</env:Body>
```

```
</env:Envelope>
```

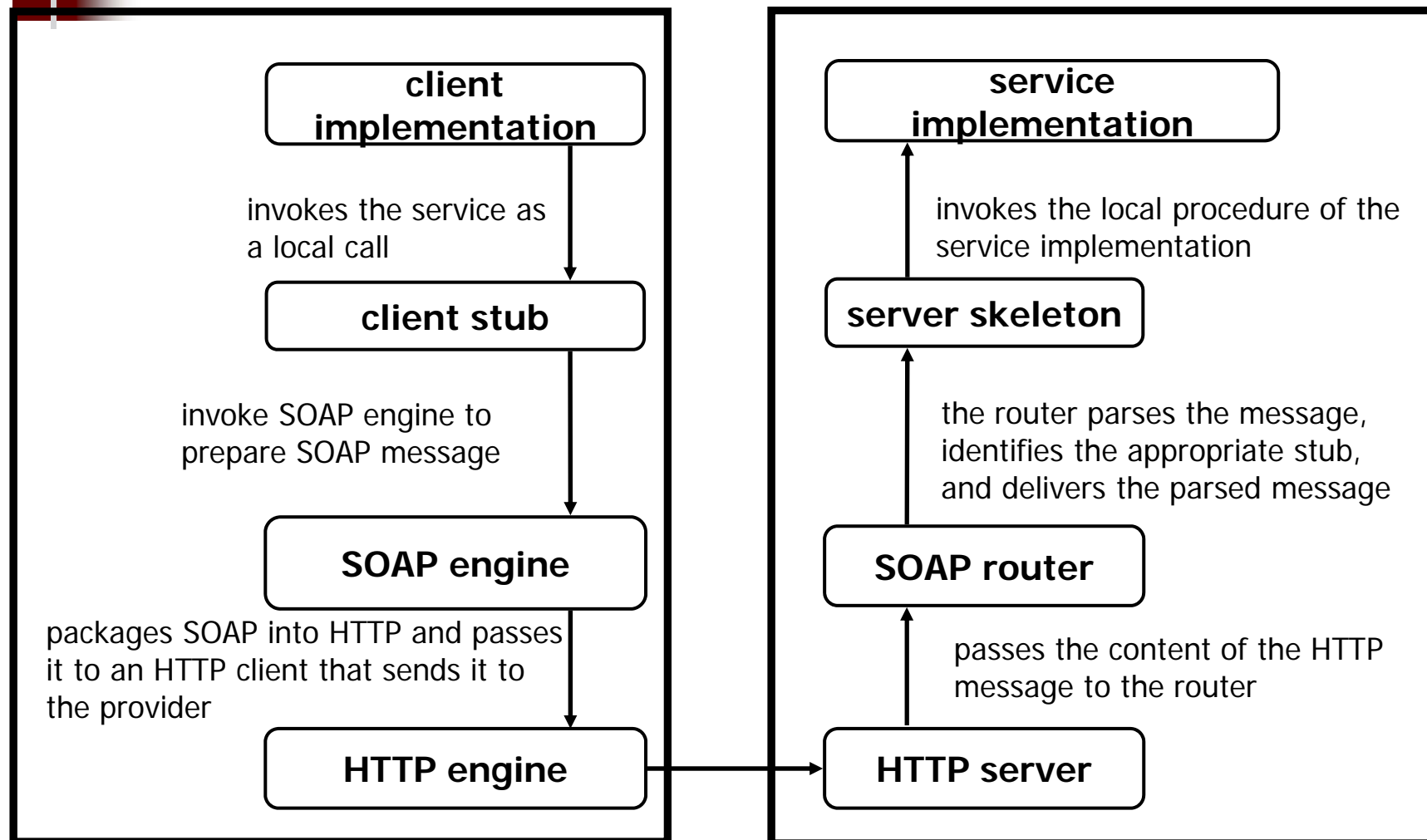
envelope

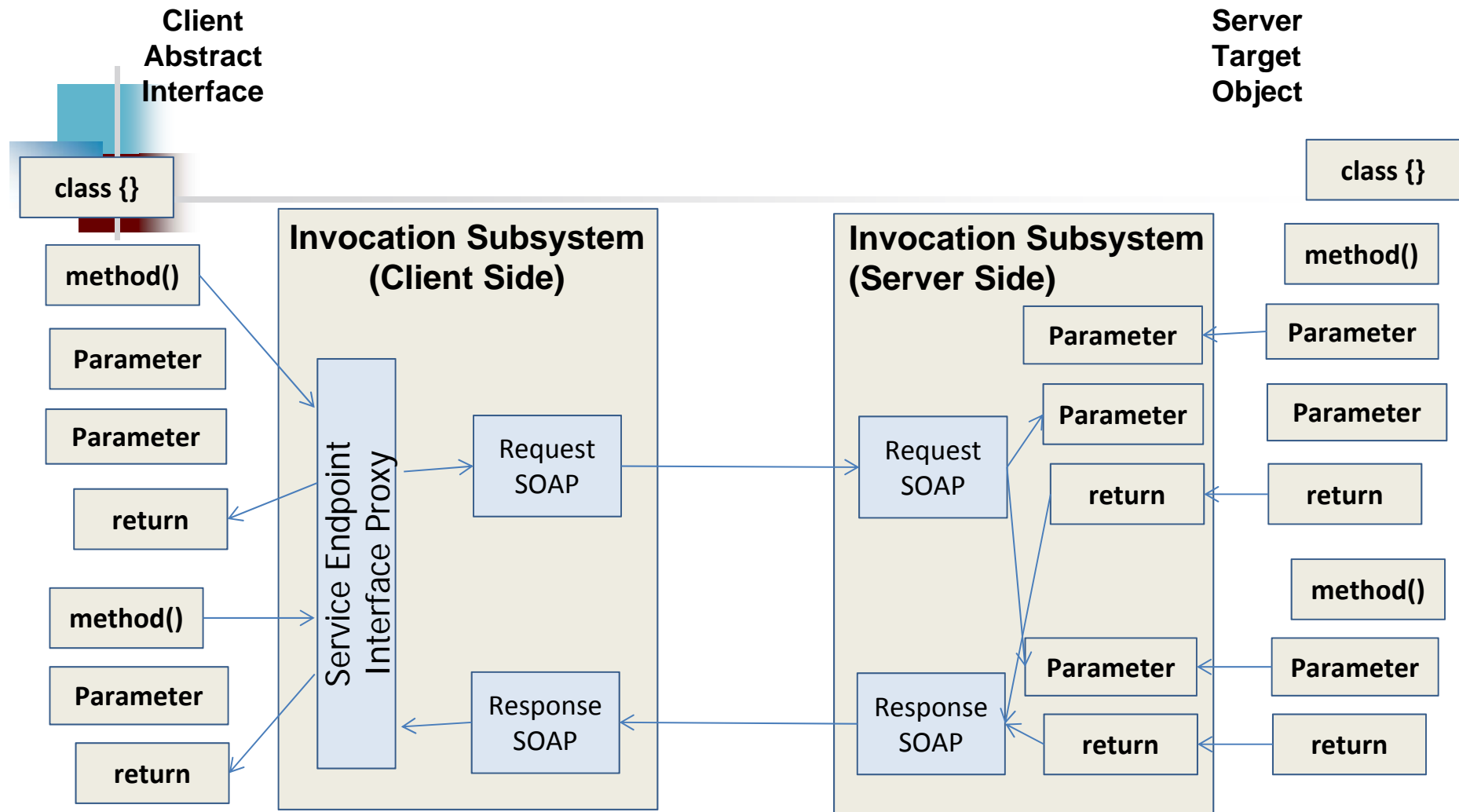
header

blocks

body

Funcionamiento de SOAP



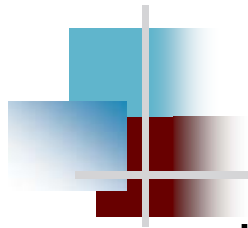


Synchronization – Serialization -- Transmission ↔ Transmission – Serialization -- Synchronization

Method
Invocation

SOAP Message
Exchange

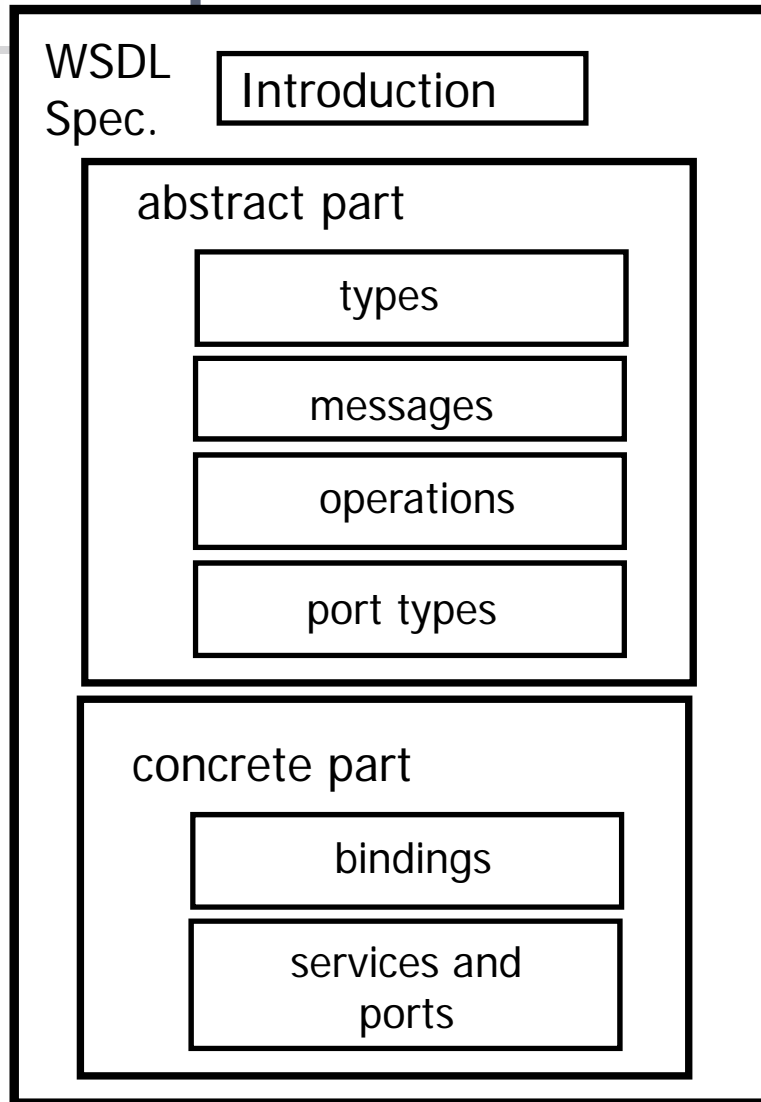
Method
Invocation



WSDL

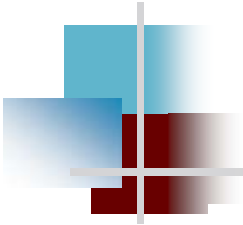
- La interfaz del Servicio Web está definida en una descripción escrita con WSDL.
- La especificación WSDL define:
 - Qué operaciones soporta el servicio y el formato en que se envían y se reciben los mensajes del servicio Web
 - Cómo se accede al servicio
 - Dónde está localizado el servicio.
 - Esto se expresa generalmente como un URI (identificador uniforme de recursos)

Estructura de una Especificación WSDL



- Declaraciones de espacios XML
- Declaraciones de tipos, interfaces y mensajes (las operaciones del servicio)
- Ligaduras y puertos

Definición de tipos



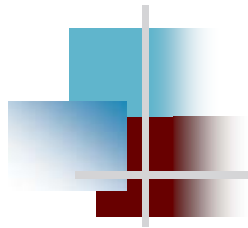
```
<types>
  <xs: schema targetNamespace = "http://.../weathns"
    xmlns: weathns = "http://.../weathns" >

    <xs:element name = "MaxMinTemp" type = "mmtrec" />
    <xs: element name = "InDataFault" type = "errmess" />

    <xs: complexType name = "pdrec"
      <xs: sequence>
        <xs:element name = "town" type = "xs:string"/>
        <xs:element name = "country" type = "xs:string"/>
        <xs:element name = "day" type = "xs:date" />
      </xs:complexType>

    ...

  </schema>
</types>
```



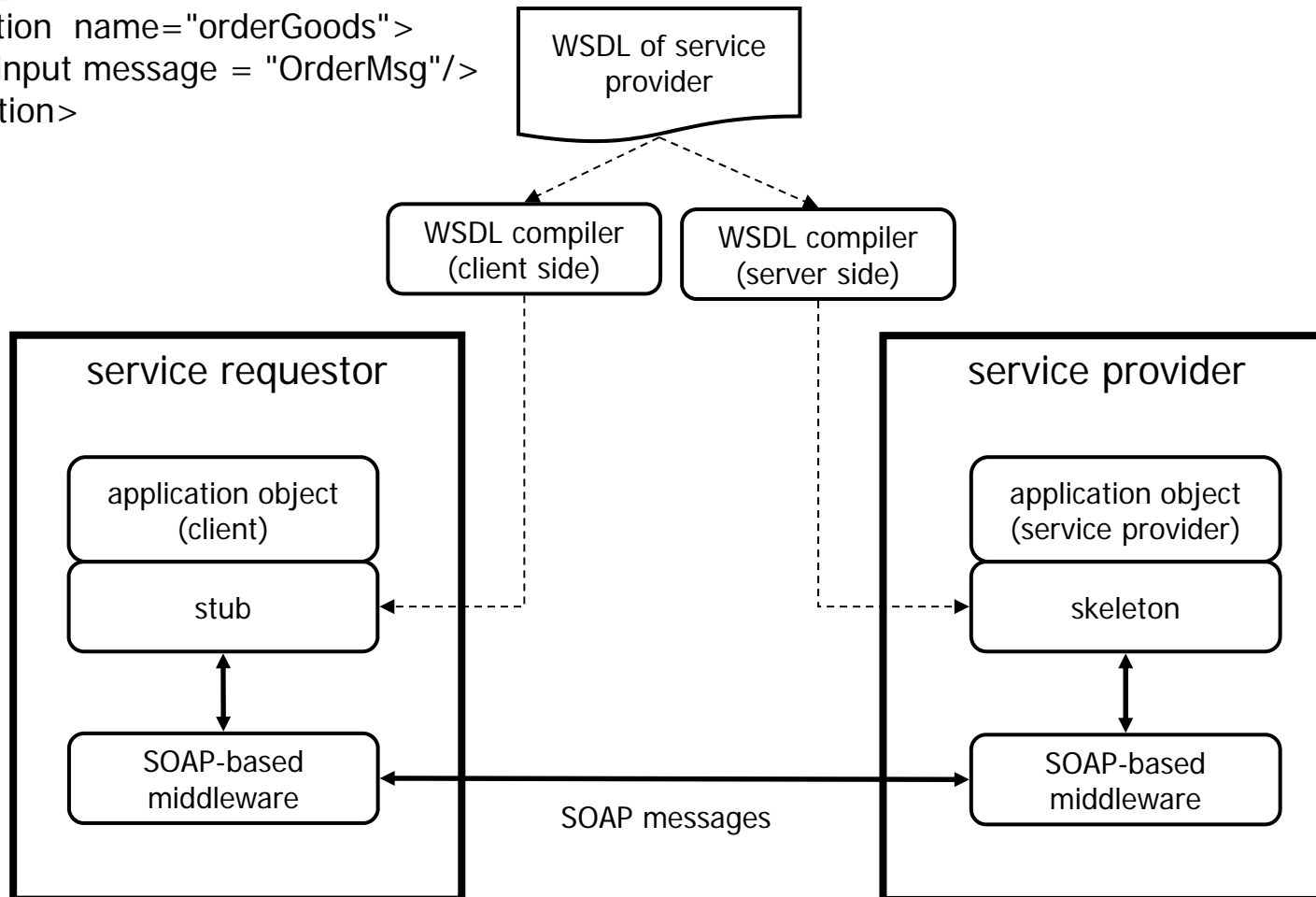
Interfaz y operaciones:

- Devuelve las temperaturas máxima y mínima

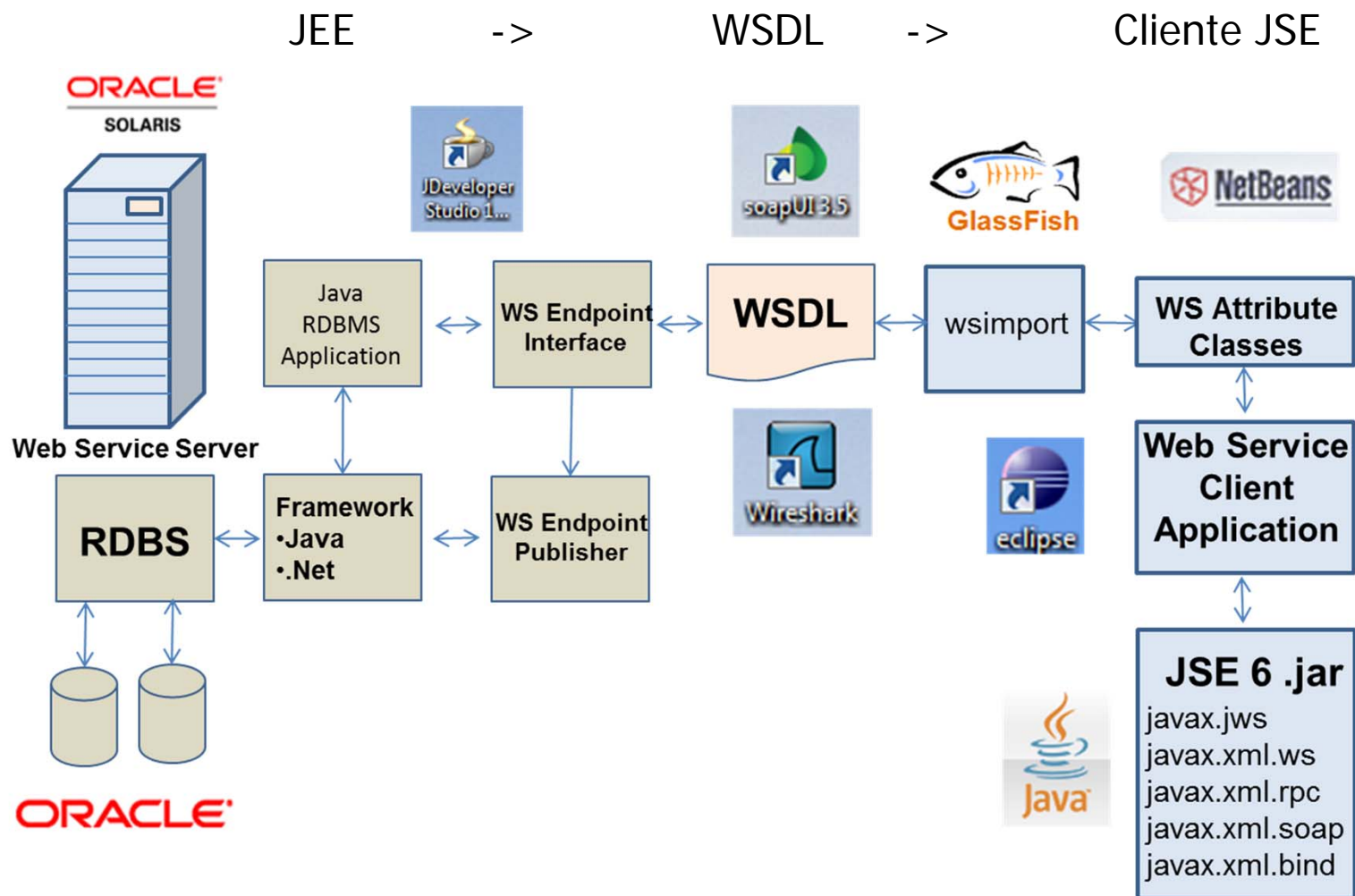
```
<interface name = "weatherInfo" >  
  <operation name = "getMaxMinTemps" pattern = "wsdl:ns: in-out">  
    <input messageLabel = "In" element = "weathns: PlaceAndDate" />  
    <output messageLabel = "Out" element = "weathns:MaxMinTemp" />  
    <outfault messageLabel = "Out" element = "weathns:InDataFault" />  
  </operation>  
</interface>
```

WDSL

```
<operation name="orderGoods">  
  <input message = "OrderMsg"/>  
</operation>
```



La "cadena de montaje"





Implementación del Servicio mediante anotaciones Java

```
public import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService(serviceName = "wsServicioWeb")
public class wsServicioWeb {
    @WebMethod(operationName = "opFactorial")
    public int opFactorial(@WebParam(name = "num") int num) {
        int i; int fact=1;
        for(i=2;i<=num;i++)
            fact*=i;
        return fact;
    }
}
```

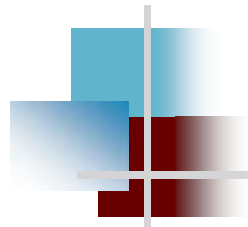
Definición del servicio Web

```
<definitions targetNamespace="http://servicios/" name="wsWebService" ...
<types>
  <xsd:schema>
    <xsd:import namespace="http://servicios/"
      schemaLocation="WsWebService_schema1.xsd"/>
  </xsd:schema>
</types>
<message name="opFactorial">
  <part name="parameters" element="tns:opFactorial"/>
</message>
<message name="opFactorialResponse">
  <part name="parameters" element="tns:opFactorialResponse"/>
</message>
<portType name="wsWebService">
  <operation name="opFactorial">
    <input wsam:Action="http://servicios/wsWebService/opFactorialRequest"
      message="tns:opFactorial"/>
    <output wsam:Action="http://servicios/wsWebService/opFactorialResponse"
      message="tns:opFactorialResponse"/>
  </operation>
</portType>
```


Definición del servicio Web

```
<binding name="wsWebServicePortBinding" type="tns:wsWebService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <operation name="opFactorial">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
<service name="wsWebService">
  <port name="wsWebServicePort" binding="tns:wsWebServicePortBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
  </port>
</service>
```

http://localhost:8080/WebApplication1/wsWebService?Tester



Test:

Method parameter(s)

Type	Value
int	4

Method returned

int : "24"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:opFactorial xmlns:ns2="http://servicios/">
      <num>4</num>
    </ns2:opFactorial>
  </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:opFactorialResponse xmlns:ns2="http://servicios/">
      <return>24</return>
    </ns2:opFactorialResponse>
  </S:Body>
</S:Envelope>
```



Cliente del servicio

```
public class JavaSEws {  
    ...  
    private static int devFactorial(int numFact) {  
        wsServicioWeb_Service service = new wsServicioWeb_Service();  
        wsServicioWeb port = service.getWsServicioWebPort();  
        return port.devFactorial(numFact);  
    }  
}
```

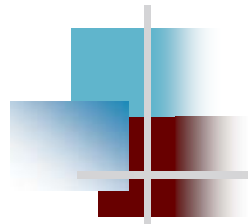
■ En un Servlet:

```
@WebServiceRef(wsdlLocation = "META-INF/wsdl/  
    localhost_8080/helloservice/HelloService.wsdl")  
private static HelloService service;  
...  
    helloservice.endpoint.Hello port = service.getHelloPort();  
    return port.sayHello(arg0);
```



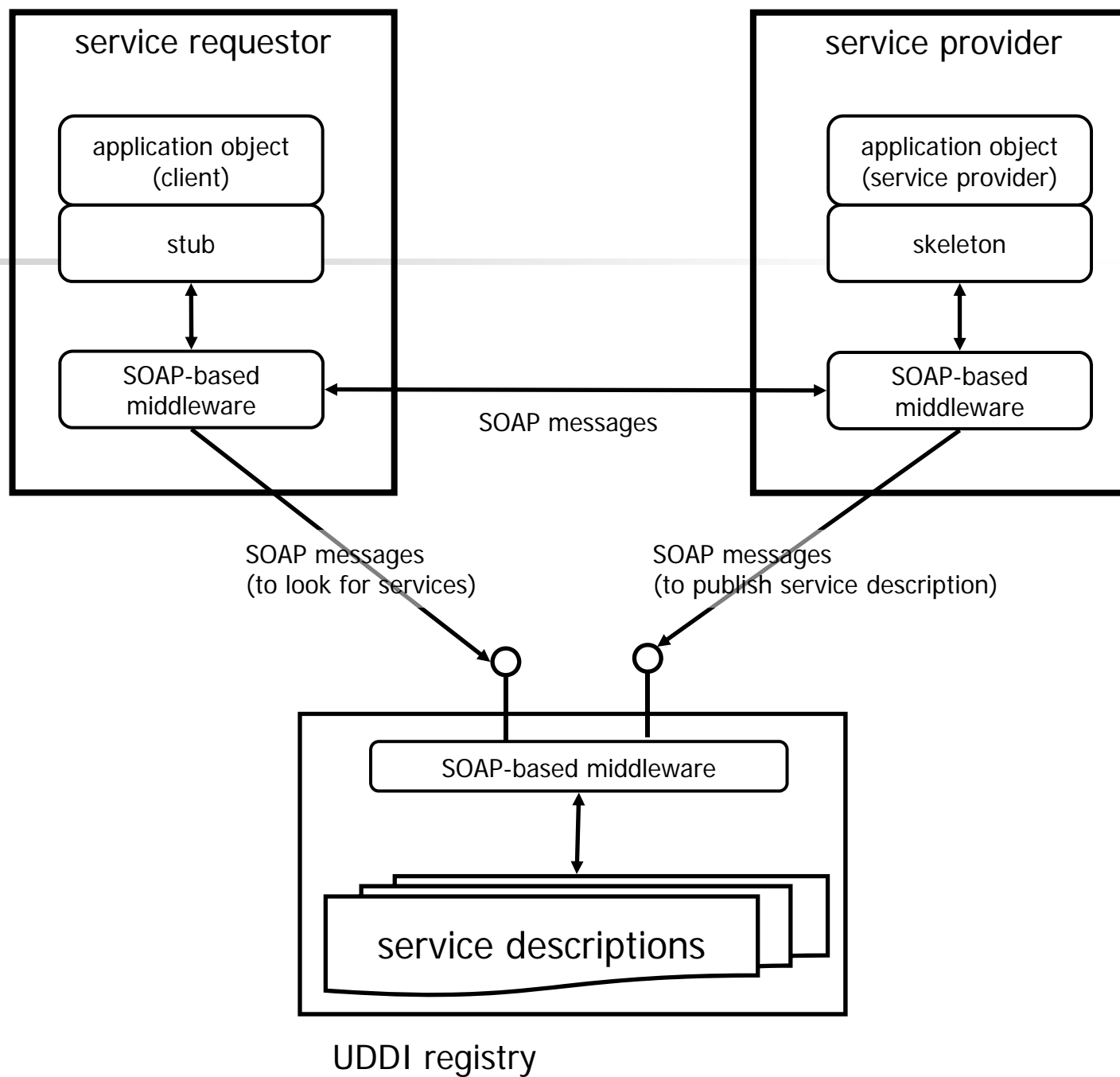
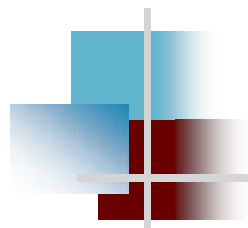
EJBs como Servicios Web

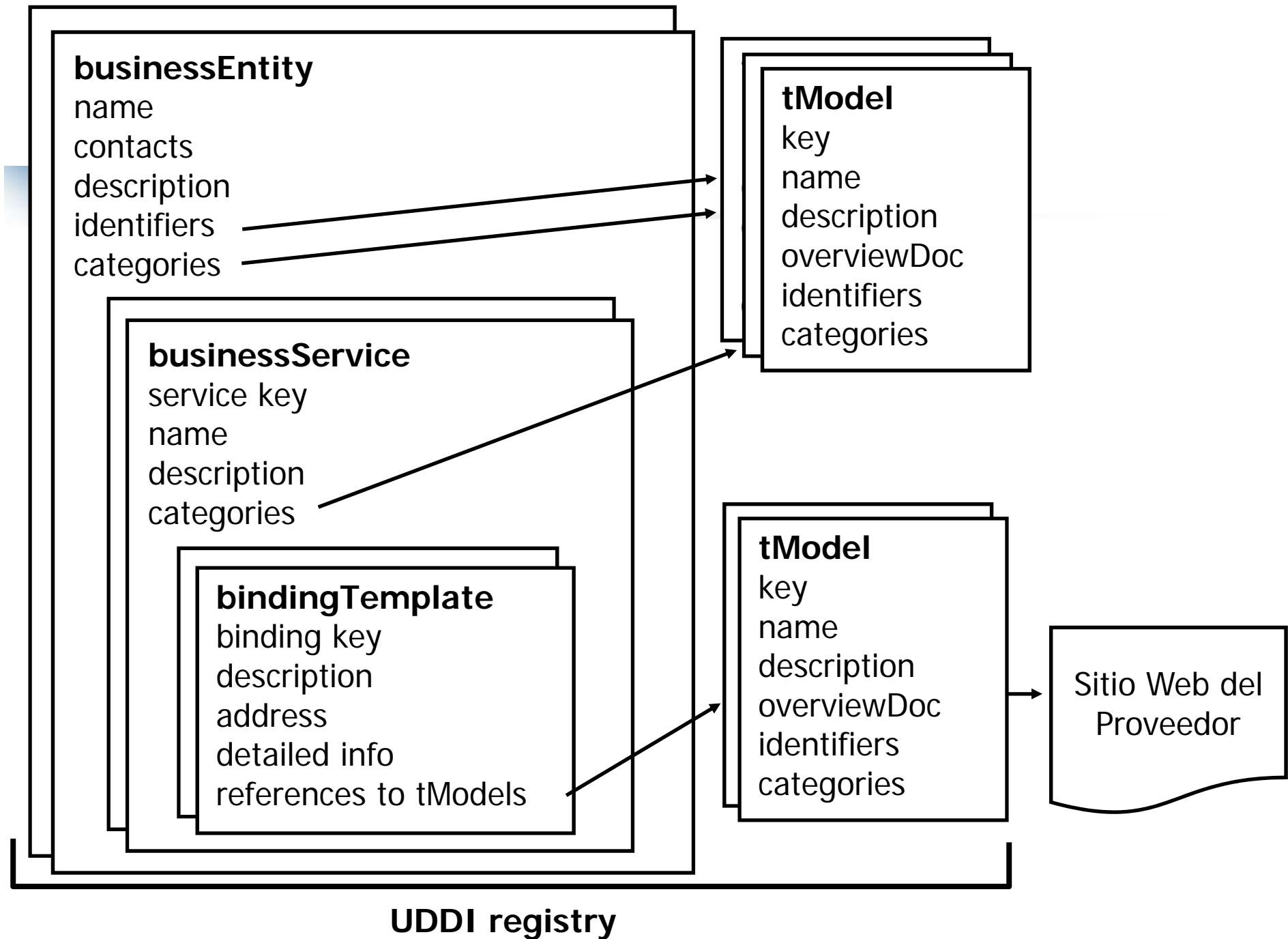
```
@WebService(serviceName = "NewWebService")
@Stateless()
public class NewWebService {
    @EJB
    private NewSessionBeanLocal ejbRef;
    /**
     * Web service operation
     */
    @WebMethod(operationName = "opFactorial")
    public int opFactorial(@WebParam(name = "num") int num) {
        return ejbRef.factorial(num);
    }
}
```



Descripción UDDI

- UDDI (Universal Description Discovery and Integration) define la especificación para poder descubrir la existencia de un servicio:
 - Detalles de la empresa que presta el servicio
 - Una descripción informal de la funcionalidad que proporciona el servicio.
 - Información sobre dónde encontrar la especificación WSDL del servicio
 - Información de suscripciones que permitan a los usuarios registrarse para tener mejoras del servicio

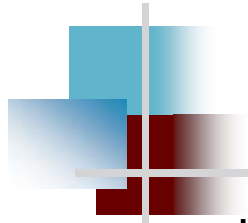






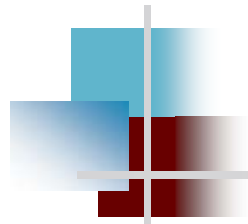
5.3. Servicios RESTful

- Principios
- Detalles de implementación
- REST y SOAP



Servicios RESTful

- Los estándares de Servicios Web han sido criticados como pesados e ineficientes
- REST (Representational State Transfer) es un estilo arquitectónico basado en la transferencia de la representación de los recursos desde un servidor a un cliente
- Este estilo es más simple que SOAP / WSDL
- Los Servicios RESTful implican una sobrecarga menor y muchas organizaciones los utilizan en sistemas que no dependen de servicios externos.
 - Ahora están generalizados (Google, Amazon...)

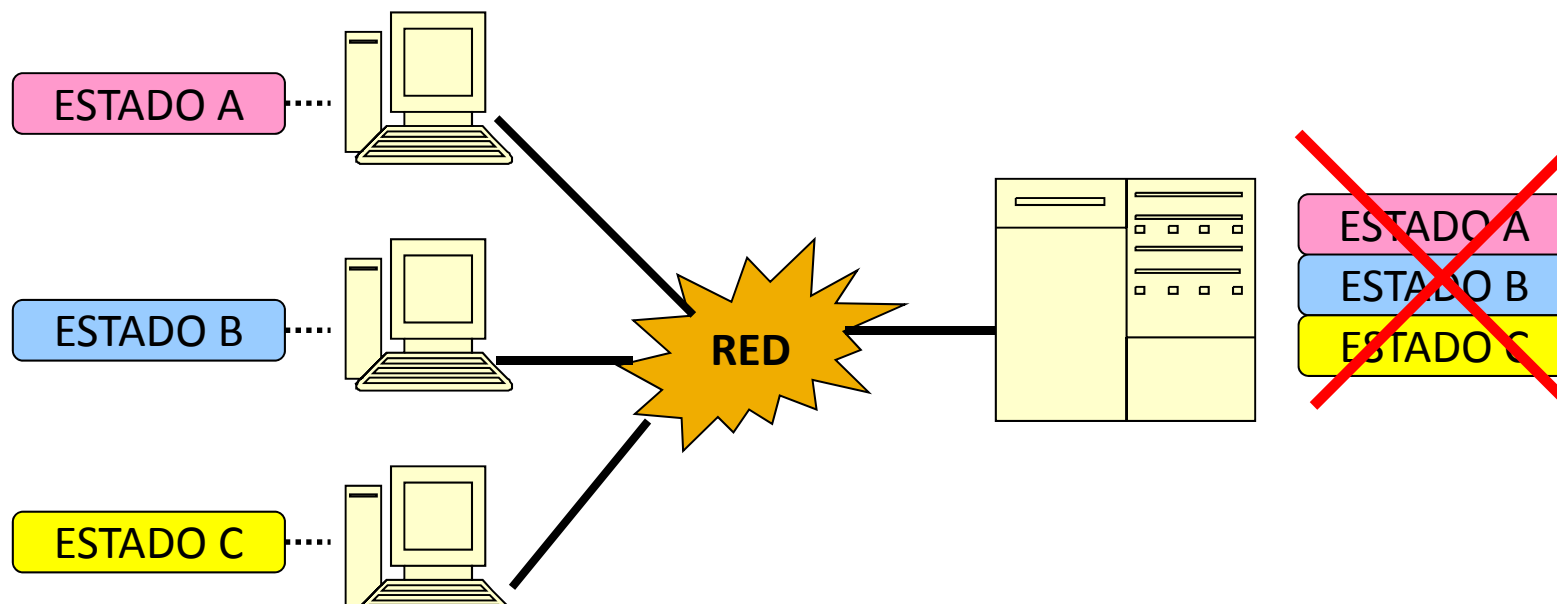


Principios de REST

- Un protocolo cliente/servidor **sin estado**
- Una sintaxis universal para **identificar los recursos**, su URI.
- Un **conjunto de operaciones** bien definidas que se aplican a todos los recursos de información(en HTTP: POST, GET, PUT y DELETE)
- El uso de **hiper-enlaces**, tanto para la información como para las transiciones de estado de la aplicación

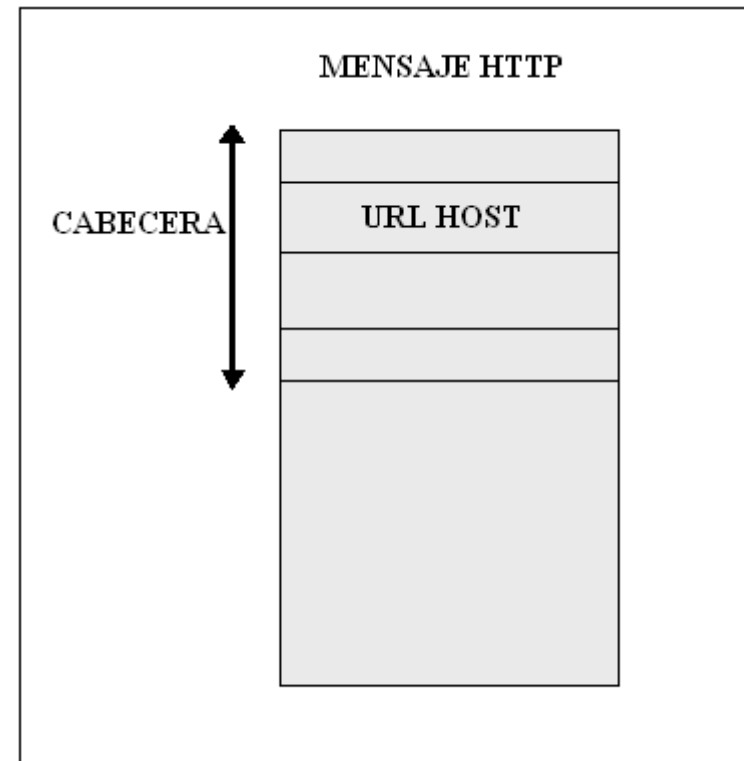
Principios de REST

- Un protocolo cliente/servidor, sin estado y basado en capas
- Cada mensaje contiene la información necesaria para comprender la petición (mensajes autocontenidos, como HTTP)



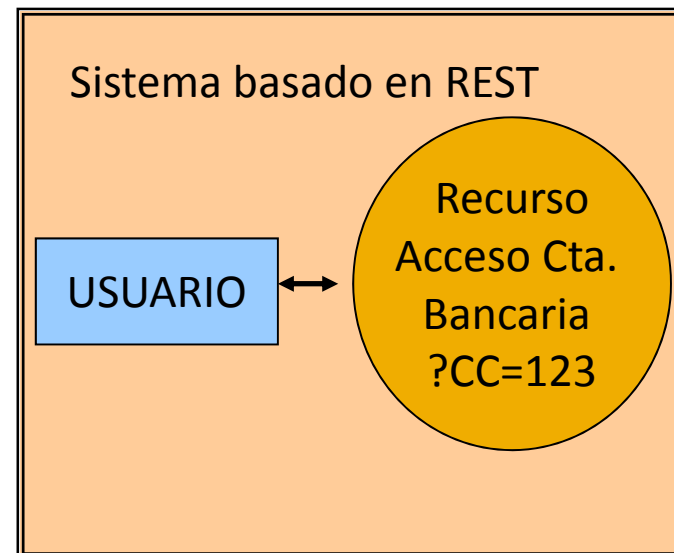
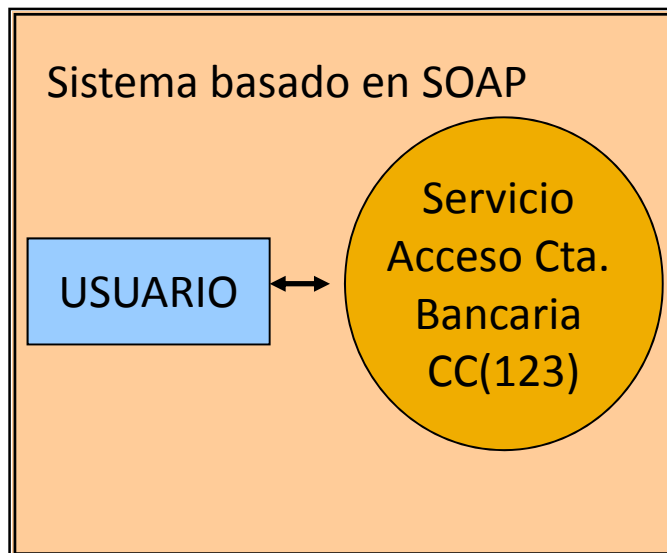
Mensajes Auto-contenidos

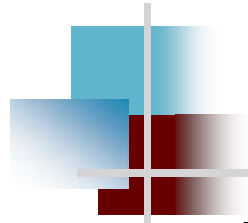
- Toda la información necesaria para procesar el mensaje se encuentra en el propio mensaje.
- Usa HTTP como protocolo de aplicación.



Recursos y no métodos

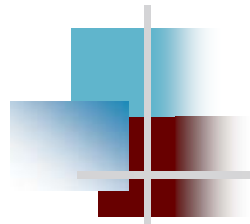
- El estado y la funcionalidad de las aplicaciones se divide en recursos
 - REST es orientado a recursos y no a métodos
 - No se accede directamente a los recursos, sino a representaciones (txt, xml, html, jpg, Json ...) de los mismos





Recursos y transferencia

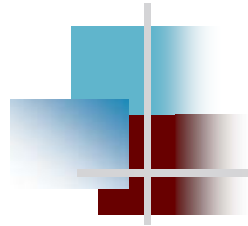
- Una solicitud REST se envía a un recurso, el URI de una máquina.
- Si la solicitud tiene éxito el cliente recibe una representación del recurso de un cierto tipo (MIME, por ejemplo texto o html, pero también imágenes)
- Esta representación es lo que se transfiere e informa del estado del recurso de alguna manera



Principios de REST

- Todo recurso es identificado de forma única global mediante una sintaxis convenida.
 - Se identifican mediante URIs (Uniform Resource Identifier).
 - Conjunto potencialmente infinito de recursos.

- Además, todos los recursos comparten una interfaz uniforme formada por:
 - Conjunto de operaciones limitado para transferencia de estado: GET, POSTS...
 - Conjunto limitado de tipos de contenidos
 - En HTTP se identifican mediante tipos MIME: XML , HTML, distintos tipos de imágenes, pdfs, ...



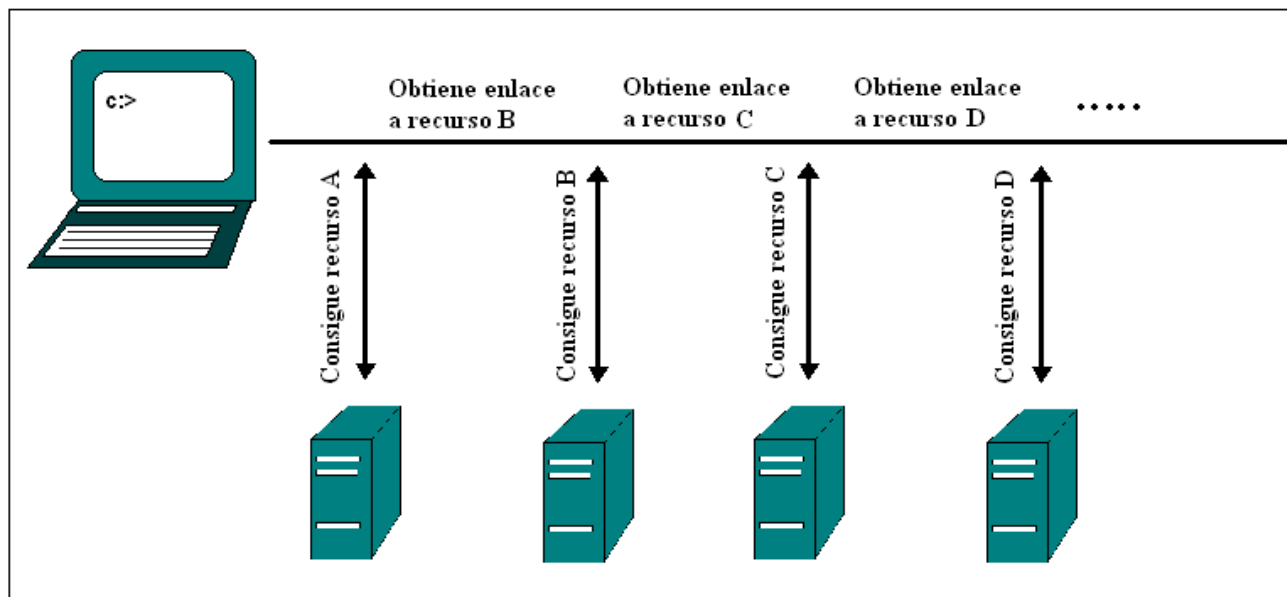
Métodos de REST

- Usa los métodos de HTTP
- Cumple el principio de interfaz uniforme

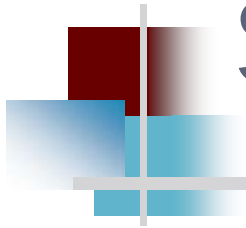
MÉTODO	FUNCIÓN
GET	Solicitar recurso
POST	Crear recurso nuevo
PUT	Actualizar o modificar recurso
DELETE	Borrar recurso

Principios de REST

- Uso de hiper-enlaces, tanto para la información de la aplicación como para las transiciones de estado de la aplicación.
- A través de sucesivas peticiones de recursos cambia el estado de la aplicación.



“Niveles de madurez” de Servicios REST





Nivel 0: "Plain Old RPC"

Un URI, un método HTTP

Natural language example: Buying food at a drive-thru. **Alice:** How much is a Coke? **Bob:** \$2 cash, \$2.25 credit. **Alice:** I'll take one Coke. Here's \$2 cash. **Bob:** Here's your Coke.

POST /drinkService

```
<checkPrice brand = "Coca Cola Classic" size = "20oz"/>
<checkPriceInfo>
  <price = "2" paymentType = "cash"/>
  <price = "2.25" paymentType = "credit"/>
</checkPriceInfo>
```



Nivel 1: Recursos

Un URI para cada recurso

Natural language example: Buying food at a counter

Alice: What drinks do you have? **Bob:** Coke for \$2, Pepsi for \$1.85, Sprite for \$2. **Alice:** I want a Sprite. Here's \$2. **Bob:** Here's your Sprite.

POST /drinks

<checkPrice />

<checkPriceltem>

<brand = "coke" price = "2"/>

<brand = "pepsi" price = "1.85"/>

<brand = "sprite" price = "2"/>

</checkPriceltem>

POST /drinks/sprite

<buyItem paymentID = "2343b23930932bfd90ac4"/>



Nivel 2: Múltiples Verbos

Múltiple URIs, múltiples métodos HTTP

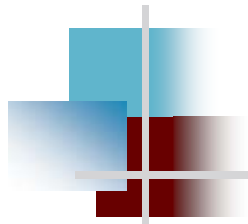
Natural language example: Entering a pub for the first time **Alice:** What can I do here? I'm legally allowed to drink. **Bob:** You can order a drink for \$3, food for \$6, or play darts for \$1. **Alice:** I'll take darts. Here is \$1. **Bob:** Here is a set of darts.

GET /bobspub/menu?drinkingAge=legal

```
<pubOfferings>
  <offering itemid="43634" item = "drink" price = "3"/>
  <offering itemid="43635" item = "food" price = "6"/>
  <offering itemid="43637" item = "darts" price = "1"/>
</pubOfferings>
```

POST /bobspub/menu/43637

```
<paymentInfo>
  <accountName = "alice1234"/>  <amount = "1"/>
</paymentInfo>
```



Nivel 3: Hypermedia

Recursos que se autoexplican

Natural language example: Transactions at a bank

Alice: How can I deposit my gold? **Bob:** You can put gold into your safe deposit box #23438aa40fd3 or convert it to cash and deposit the cash in your checking account #9909n339.

Alice: Here is \$5000 in gold for safe deposit box #23438aa40fd3.

Bob: OK. Your safe deposit box #23438aa40fd3 now has \$5000 in gold. Also, just call me at BankNumber and have your safe deposit box number (#23438aa40fd3) ready if you want to withdraw anything.

[months later]

Alice: I'd like to withdraw \$3000 in gold from safe deposit box #23438aa40fd3.

Ted: Here's your gold. Your safe deposit box #23438aa40fd3 now has \$2000 in gold.



Nivel 3: Hypermedia

GET /accounts/alice1337

<accountsList>

<account name = "safeDepositBox" currentValue = "0">

<link rel = "deposit" uri = "/accounts/alice1337/23438aa40fd3"/>

<link rel = "withdraw" uri = "/accounts/alice1337/23438aa40fd3"/>

</account>

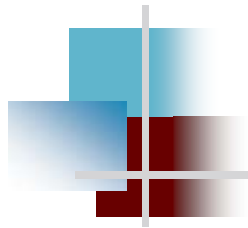
<account name = "checkingAccount" currentValue = "2000">

<link rel = "deposit" uri = "/accounts/alice1337/9909n339"/>

<link rel = "withdraw" uri = "/accounts/alice1337/9909n339"/>

</account>

</accountsList>



Nivel 3: Hypermedia

POST /accounts/alice1337/23438aa40fd3

<transactionInfo>

<deposit medium = "gold" value = "5000"/>

</transactionInfo>

HTTP/1.1 201 Created

<accountInfo>

<account name = "safeDepositBox" currentContents="gold" currentValue="5000">

<uri = "/accounts/23438aa40fd3"/>

</account>

<link rel = "deposit" uri = "/accounts/23438aa40fd3"/>

<link rel = "withdraw" uri = "/accounts/23438aa40fd3"/>

</accountInfo>

POST /accounts/alice1337/23438aa40fd3

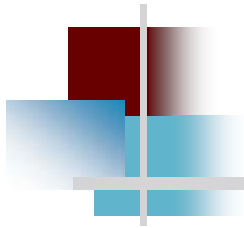
<transactionInfo>

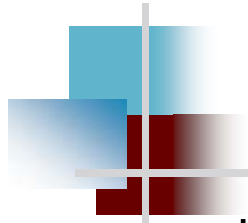
<withdraw medium = "gold" value = "3000"/>

</transactionInfo>

5.4. REST:

Detalles de implementación





Servicios RESTful: detalles

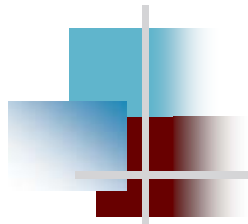
- Los servicios se modelan como recursos y permiten su acceso mediante su URI
- Los servicios responden a los métodos HTTP: GET, POST, PUT y DELETE.
- La información puede ser codificada en XML u otras representaciones como JSON (Javascript Object Notation)
- SOAP y REST no son incompatibles
- los proveedores de servicios de Internet pueden ofrecer interfaces de servicios SOAP y REST.
 - Amazon ofrece ambos y su experiencia es que la interfaz RESTful es la preferida por los desarrolladores.



JSON frente a XML

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
  </phoneNumbers>
</person>
```

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```



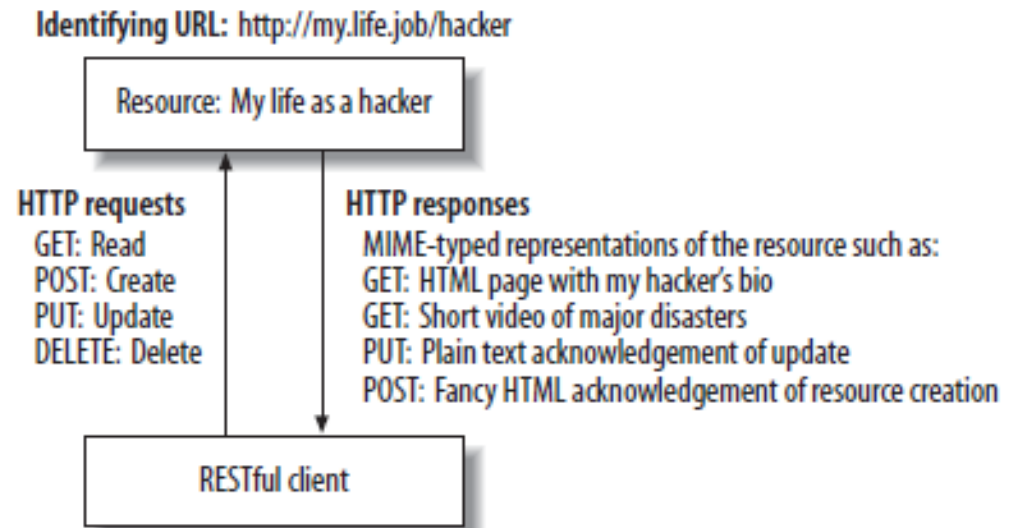
Guías

- doGet() y doPost() manejan las llamadas GET y POST
 - Ambos reciben los mismos parámetros de tipo HttpServletRequest y HttpServletResponse (no hay apenas distinción)
- Un principio básico del estilo REST es respetar los significados originales de los verbos HTTP.
 - Una solicitud GET debe estar libre de efectos secundarios (un GET es una lectura)

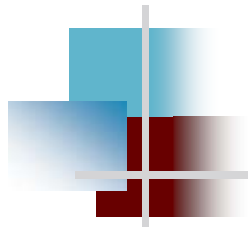
Métodos

- Los métodos HTTP se interpretan como operaciones CRUD:

- POST = Create
- GET = Read
- PUT = Update or Create
- DELETE = Delete



- Extensibilidad:
 - Una solicitud GET puede devolver una representación HTML que puede incluir hipervínculos a otros recursos, que a su vez podrían ser nuevas peticiones GET



Ejemplos típicos

HTTP verbo/URI

POST http://emp.org/emps

GET http://emp.org/emps

GET http://emp.org/emps ?id=27

PUT http://emp.org/emps

DELETE http://emp.org/emps

DELETE http://emp.org/emps ?id=27

Significado CRUD

Crear un nuevo empleado

Leer una lista con todos los empleados

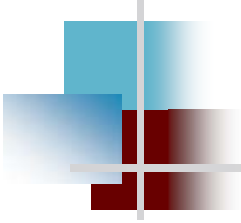
Leer el empleado con id=27

Actualizar la lista de empleados

Borrar la lista de empleados

Borrar el empleado con id=27

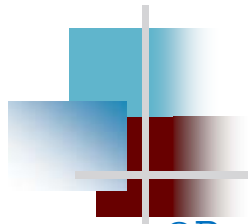
- Los datos se incluyen en el objeto `HttpServletRequest`



REST y Anotaciones Java: Hola Mundo!

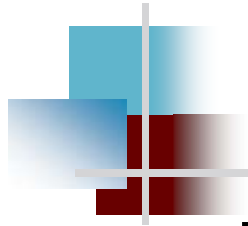
```
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.ProduceMime;

@Path("/helloRest")
public class HelloRest {
    @GET
    @ProduceMime("text/html")
    public String sayHello() {
        return "<html><body><h1>Hello from
        Jersey!</body></h1></html>";
    }
}
```



Texto plano

```
@Path("/helloworld")
public class HelloWorldResource {
    // el método procesa peticiones GET de tipo String
    @GET
    @Produces("text/plain")
    public String getMessage() {
        return "Hello World";
    }
}
```

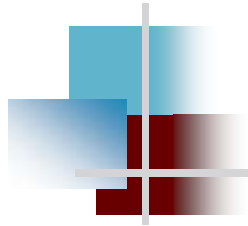



HolaMundo!

- Este servicio Web se ejecuta simplemente colocando la URL en el navegador.

<http://localhost:8080/HelloWorld/resources/helloRest>

- Si hay que pasar parámetros, se pasan a través de la URI por medio de los verbos HTTP.



Parámetros de URI (@Path)

```
@Path("customer/{name}")
public class Customer {
    @GET
    String get(@PathParam("name") String name) { ... }
    @PUT
    Void put(@PathParam("name") String name, String
        value) { ... }
```



Parametros de URI

```
@Path("/products/{id}")
public class ProductResource {
    @Context
    private UriInfo context;
    /** Creates a new instance of ProductResource */
    public ProductResource() { }

    @GET
    @ProduceMime("text/plain")
    public String getProduct(@PathParam("id") int productId)
    {
        switch (productId) {
            case 1: return "A Shiny New Bike";
            case 2: return "Big Wheel";
            case 3: return "Taser: Toddler Edition";
            default: return "No such product";
        }
    }
}
```



Parametros de URI

```
import javax.ejb.Stateless;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.QueryParam;

@Path("/factorial")
@Stateless
public class Factorial {
    @GET
    public String factorial(@QueryParam("numero") long
numero) {

        return Long.toString($factorial(numero));
    }
    ...
}
```

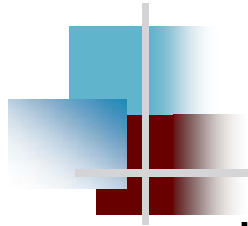


Expresiones regulares

```
@Path("/products/{ id: \\d{3} }")
public class ProductResource {
    public ProductResource() { }
    @GET
    @Produces("text/plain")
    public String
        getProductPlainText(@PathParam("id") int
            productId) {
        return "Your Product is: " + productId;
    }
}
```

- Incorrecto (devuelve estado 404):

<http://localhost:8080/jrs/resources/products/7>



JAX-RS

- La descripción de los WS con REST se puede hacer a través de WADL (Web Application Description Language),
 - archivo basado en XML que contiene las operaciones disponibles y su forma de acceso a través del protocolo HTTP.



Ejemplo 1: SOAP/REST

POST /InStock HTTP/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">

<m:GetStockPrice>

<m:StockName>IBM</m:StockName>

</m:GetStockPrice>

</soap:Body>

</soap:Envelope>

GET /stock/IBM HTTP/1.1

Host: www.example.org

Accept: application/xml



Ejemplo 2: SOAP/REST

POST /InStock HTTP/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?xml version="1.0"?

<soap:Envelope xmlns

soap:encodingStyle="

<soap:Body

<

<?xml version="1.0"?>

<order>

<

<StockName>IBM</StockName>

</soap:Body

<Quantity>50</Quantity>

</soap:Envelope>

</order>



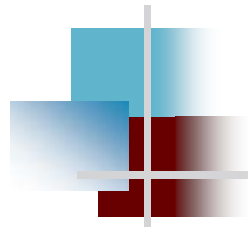
Alternativa directa

```
import javax.ws.rs.client.*;
import org.codehaus.jettison.json.*;

//conectar con el servicio y obtener un String json
Client client =ClientBuilder.newClient();
WebTarget webTarget =
    client.target("http://.../").path("cli/"+nif+".json");
String s = webTarget.request().get(String.class);
//transformar un String en objeto json
JSONObject jobject = new JSONObject(s);
nomR = (String) jobject.get("nombre");
//mejor: extraer la instancia Java del obj. json equivalente
Cliente cli=new Cliente();
ObjectMapper mapper = new ObjectMapper();
cli = mapper.readValue(s, Cliente.class); //cli.getNombre()
```

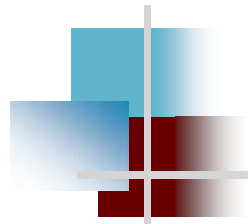


REST Y SOAP



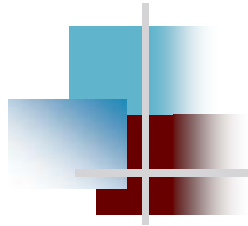
REST Y SOAP

SOAP	REST
Origen en el ámbito académico	Origen en el ámbito de las empresas
Orientado a RPC (métodos)	Orientado a recursos
Servidor almacena parte del estado	El estado se mantiene sólo en el cliente, y no se permiten las sesiones
Usa HTTP como transporte (túnel) para el paso de mensajes	Propone HTTP como nivel de aplicación



Ventajas de REST

- Mejora el tiempo de respuesta gracias al mecanismo Caché y los mensajes auto-descriptivos.
- Disminución de carga en servidor
- Mayor escalabilidad al no requerir mantenimiento de estado en el servidor
- Facilita desarrollo de clientes (menor dependencia del servidor).
- Mayor estabilidad frente a futuros cambios
 - Permite evolución independiente de los tipos de documentos al ser procesados éstos en el cliente.



Uso de REST

- Adecuado para grandes cantidades de información pública para grupos desconocidos de usuarios
 - Implementación típica (ej: Rails) MVC donde el Modelo es también un “active record”
 - Maven: REST from JPA
 - Fácil para operaciones CRUD de una sola tabla
- No adecuado para sistemas complejos, transacciones, etc



Ejemplos de Implementaciones

■ AMAZON

- Pionera en el uso de REST en 2002
- Base de datos con todos los productos que vende
- Los productos se acceden como recursos, no como métodos de búsqueda
- API disponible en associates.amazon.com
- Posible carencia, si realiza servicios más sofisticados puede que deba migrar a SOAP

■ EBAY

- Desarrolló una API REST en 2004
- Consulta de productos a través del método `GetSearchResults()`

■ OTROS: YOUTUBE, YAHOO, FLICKR, etc..

- En ocasiones siguen la arquitectura “sin querer”.



Ejemplo

- Sistema basado en SOAP

- operaciones (verbos)

- getUser()
- addUser()
- removeUser()
- updateUser()
- getLocation()
- addLocation()
- removeLocation()
- updateLocation()
- listUsers()

- Sistema REST

- recursos (nombres)

User {} Location{}

- Registro del recurso User
(accesible con HTTP GET):

```
<usuario>
<nombre>Benito Pérez</nombre>
<genero>masculino</genero>
<localizacion
href="http://www.example.org/locations/s
pain/oviedo"> Oviedo, Spain
</localizacion>
</usuario>
```



Inconvenientes SOAP

- SOAP no es transparente, apuesta por el encapsulamiento
- SOAP no dispone de un sistema de direccionamiento global
- SOAP puede derivar en agujeros de seguridad
- SOAP no aprovecha muchas de las ventajas de HTTP al usarlo solamente como túnel
- SOAP no puede hacer uso de los mecanismos Caché



Inconvenientes REST

- REST es poco flexible
- REST no está preparado para albergar Servicios Web de gran complejidad como las aplicaciones B2B
- REST falla a la hora de realizar Servicios Web que necesiten procesamiento de datos
- REST tiene grandes problemas de seguridad al no soportar el concepto de sesión