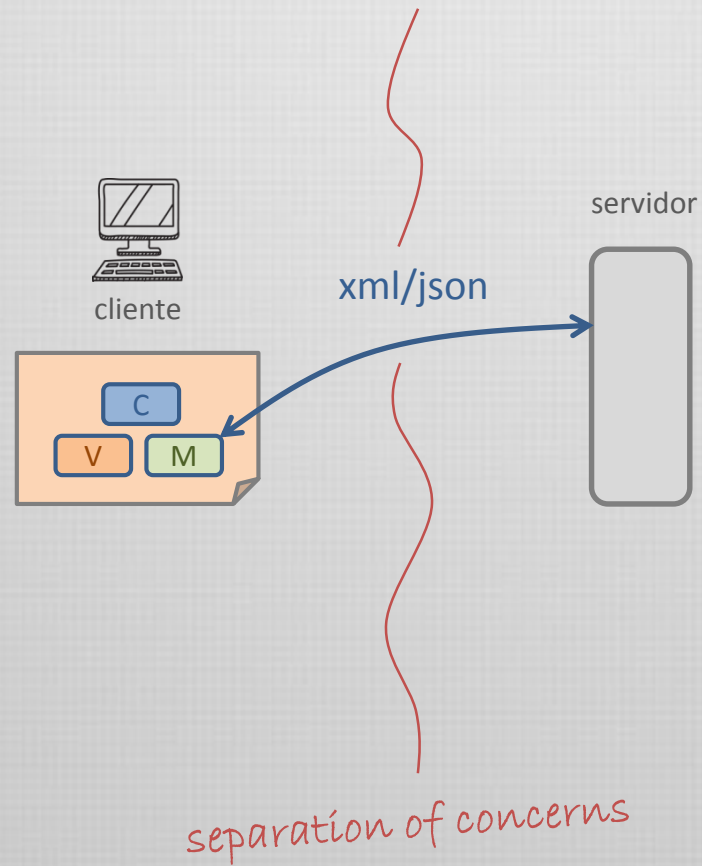


Aplicaciones Web

Diego C. Martínez

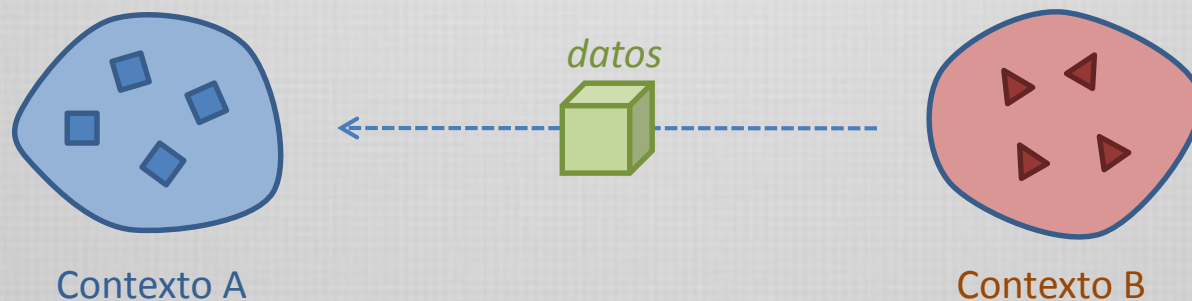
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Balance cliente - servidor



Interoperabilidad

interoperabilidad → *Habilitar el uso de información generada en otro contexto, de forma tan automatizada como sea posible.*

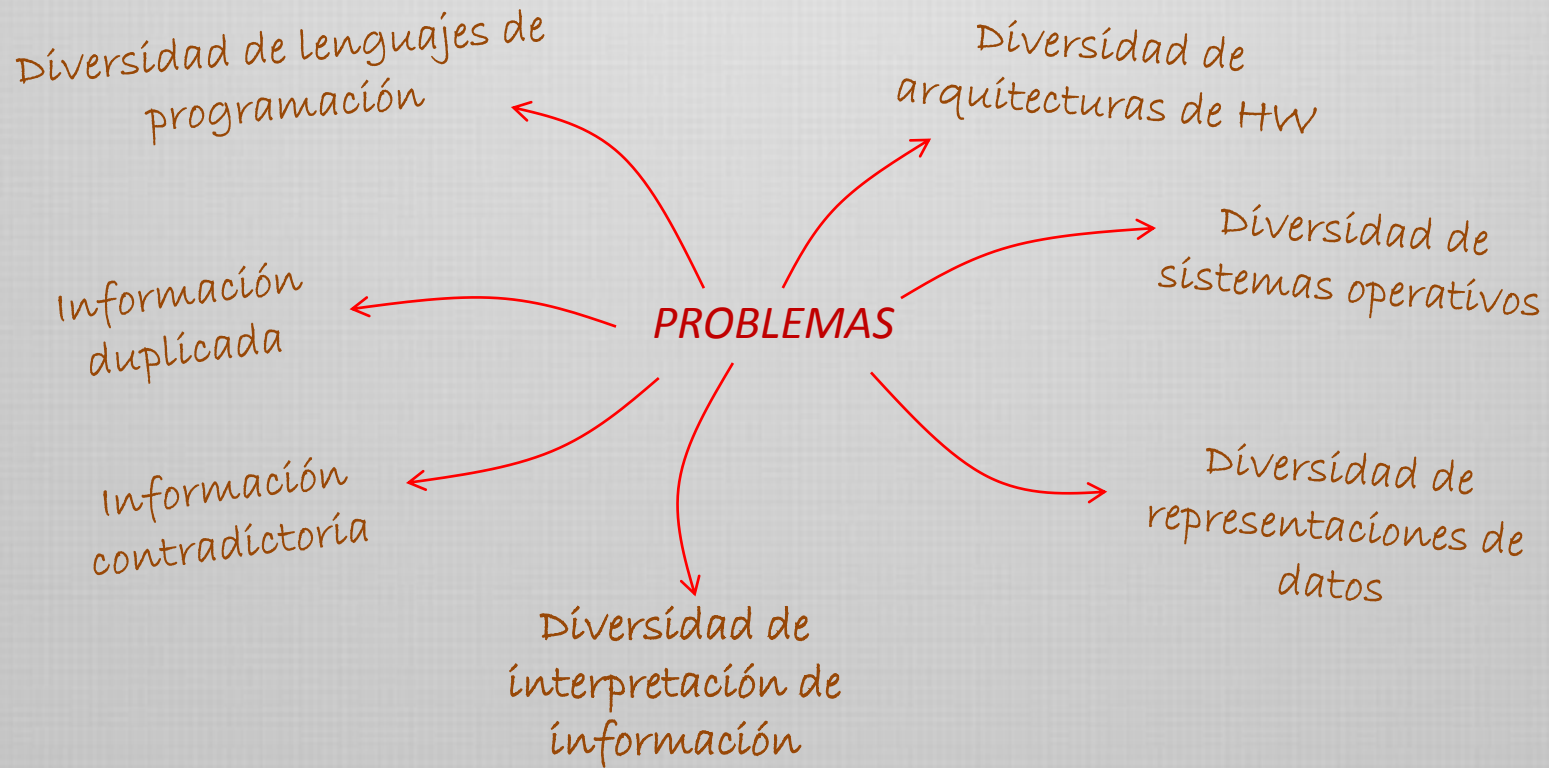


No es un concepto nuevo, ni es exclusivo de las tecnologías de información

*En la historia de la computación ha tomado varias formas,
según las tecnologías vigentes en cada época*

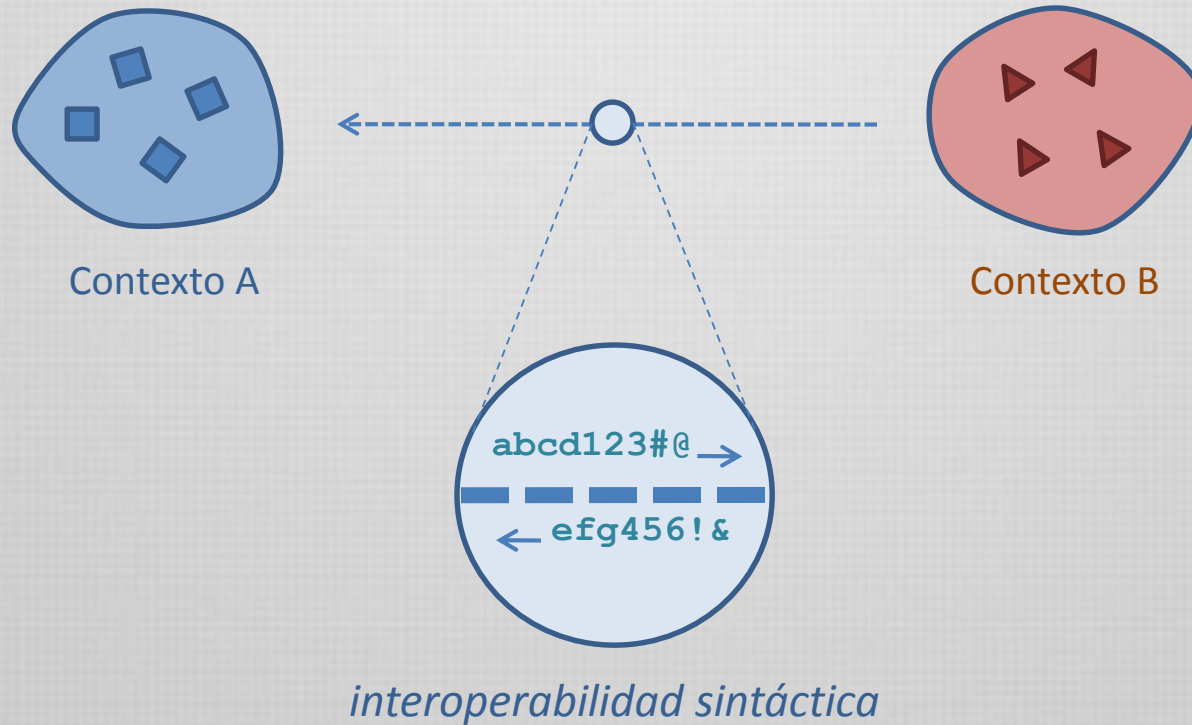
Interoperabilidad

interoperabilidad → *Habilitar el uso de información generada en otro contexto, de forma tan automatizada como sea posible.*



“architectural mismatch”

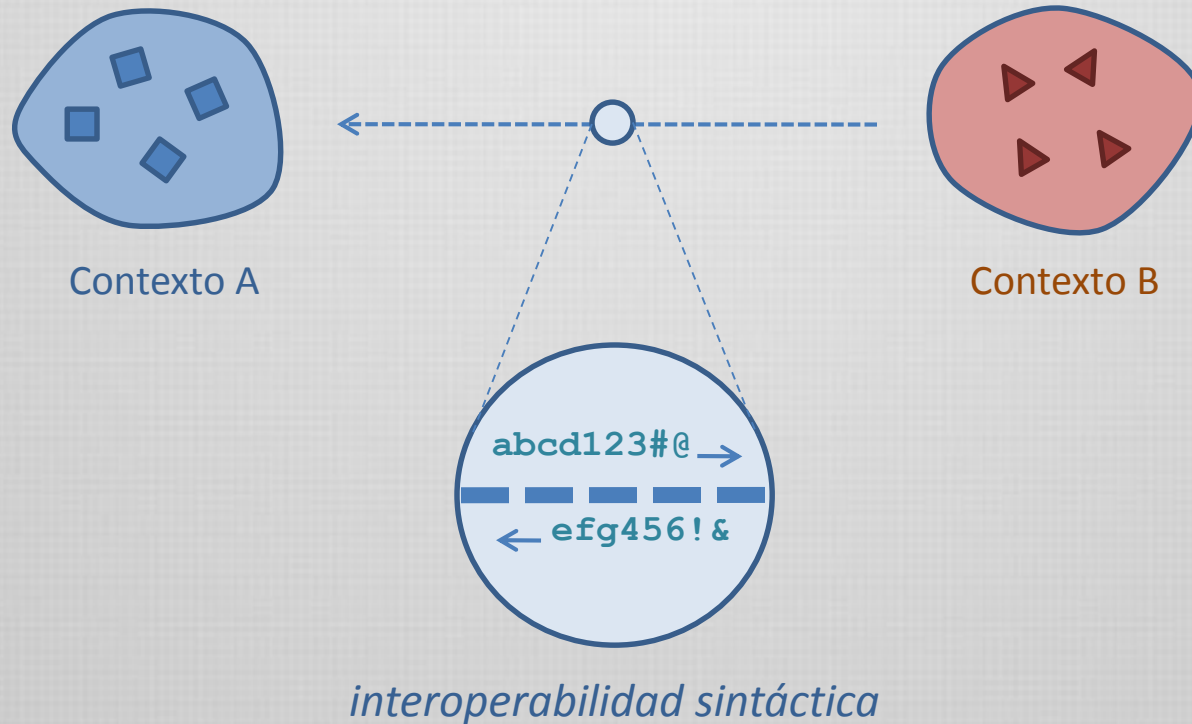
Interoperabilidad



“...technical issues of linking computer systems and services. It includes key aspects such as open interfaces, interconnection services, data integration and middleware, data presentation and exchange, accessibility and security services.”

European Interoperability Framework - EIF

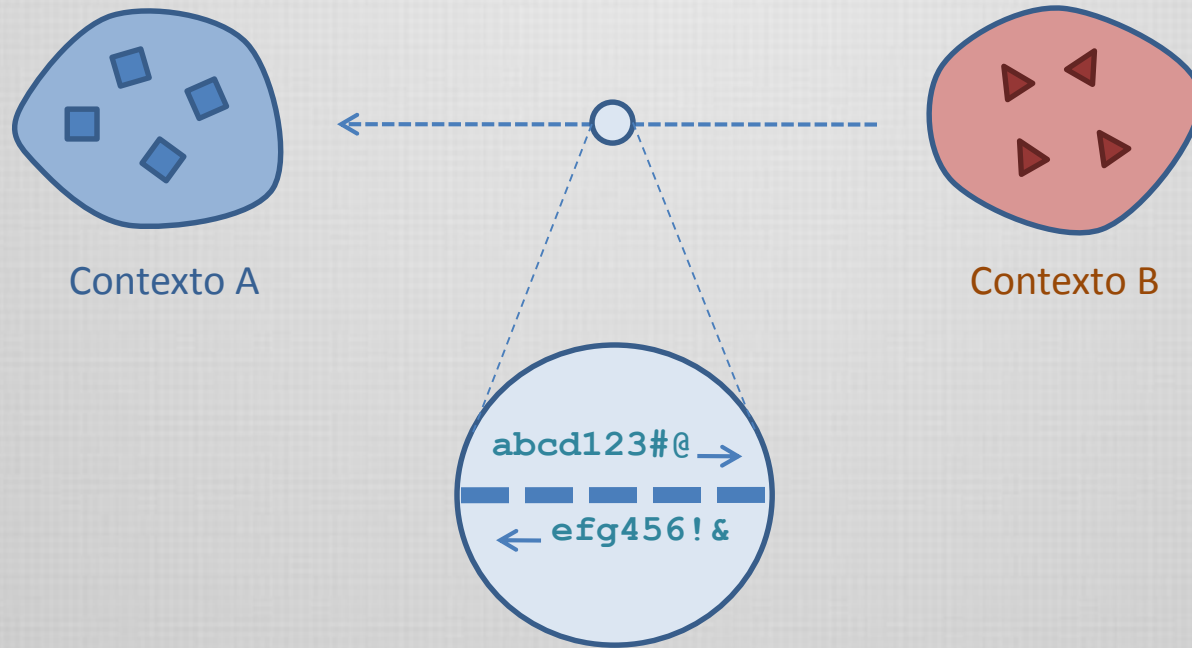
Interoperabilidad



“...is usually associated with hardware/software components, systems and platforms that enable machine-to-machine communication to take place. (...) often centered on (communication) protocols and the infrastructure needed for those protocols to operate”.

European Telecommunication Standards Institute (ETSI)

Interoperabilidad

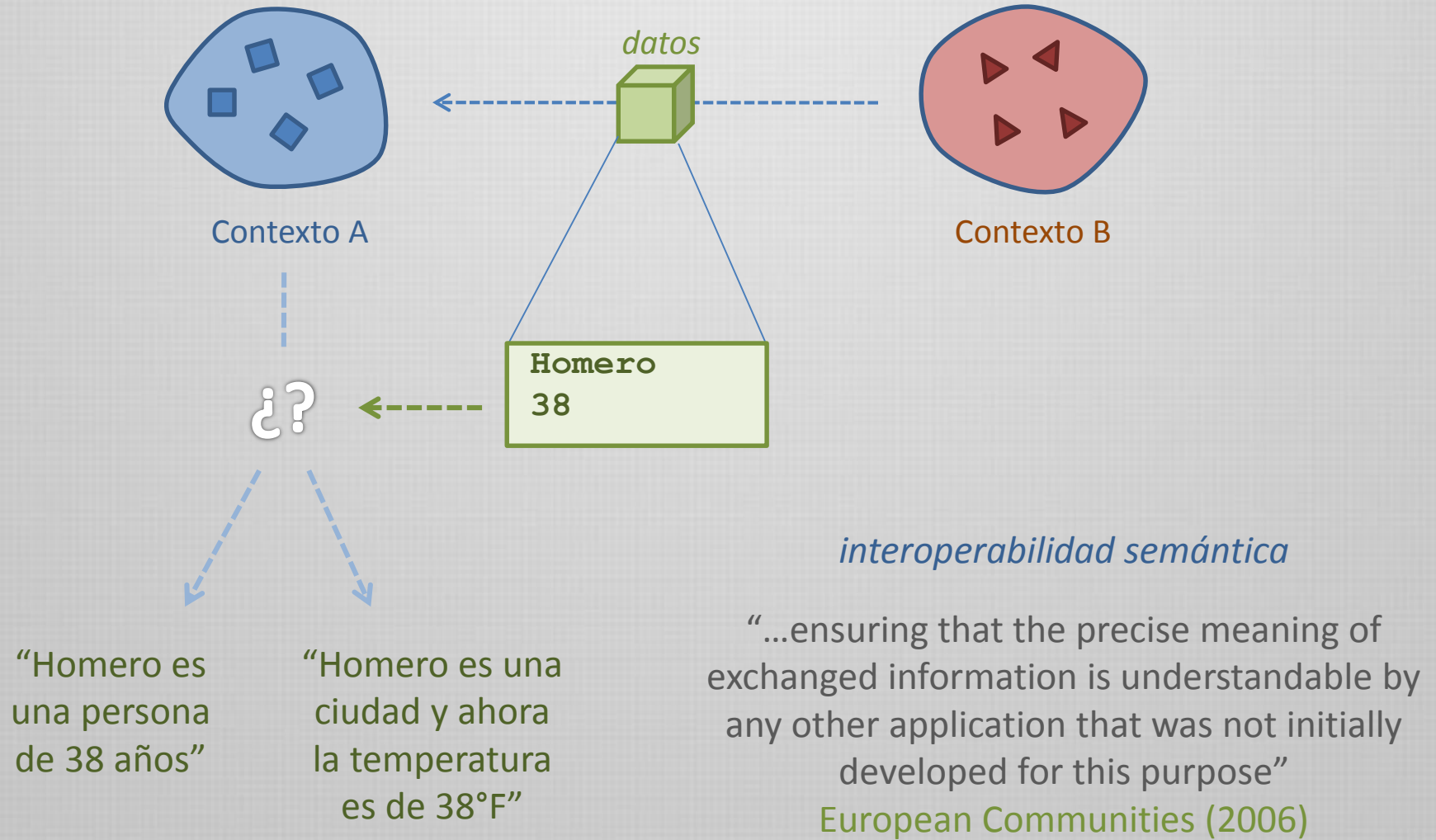


interoperabilidad sintáctica

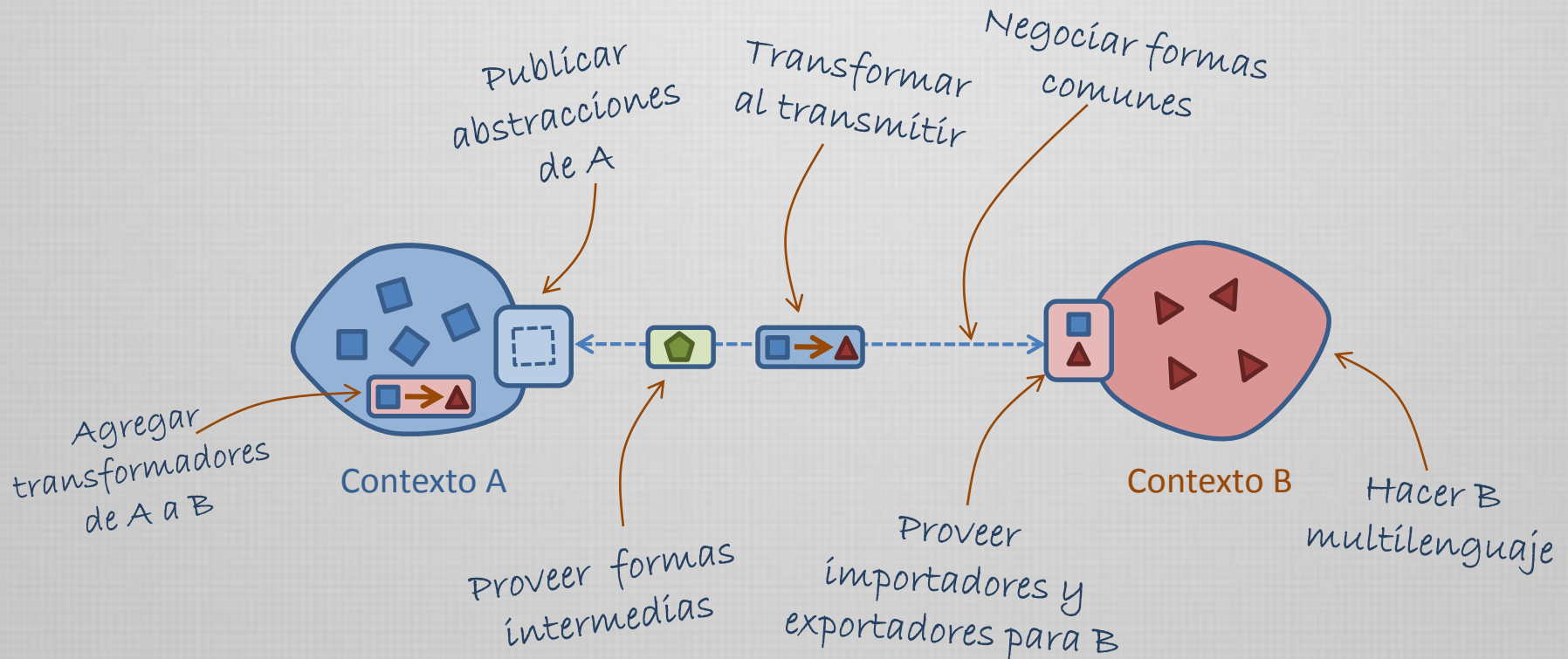
Desde la creación de [Internet](#), la interoperabilidad sintáctica no es una barrera para la operación



Interoperabilidad

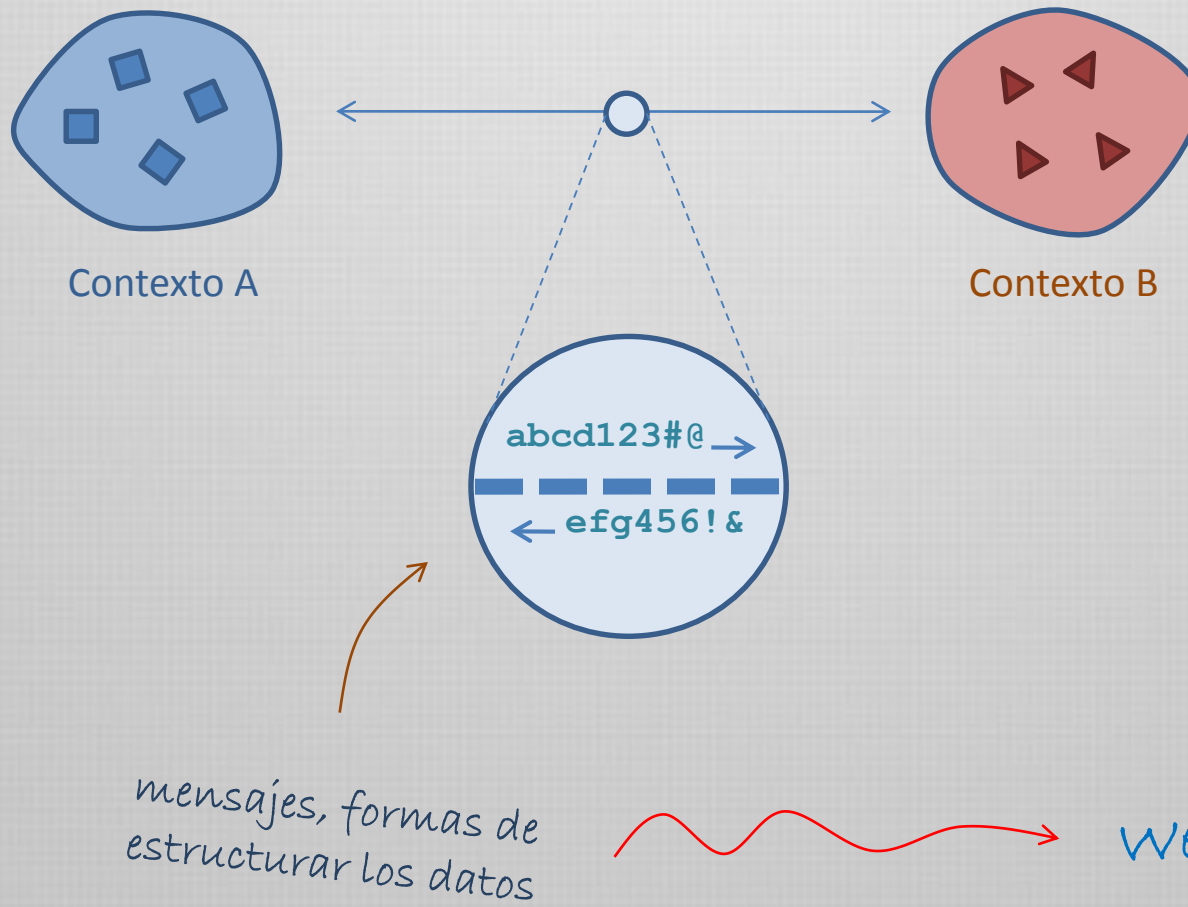


Architectural Mismatch – alternativas



Interoperabilidad sintáctica

Nosotros ya conocemos los mecanismos que permiten la interoperabilidad sintáctica



Web Service

¿qué es un *servicio web*?

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL).

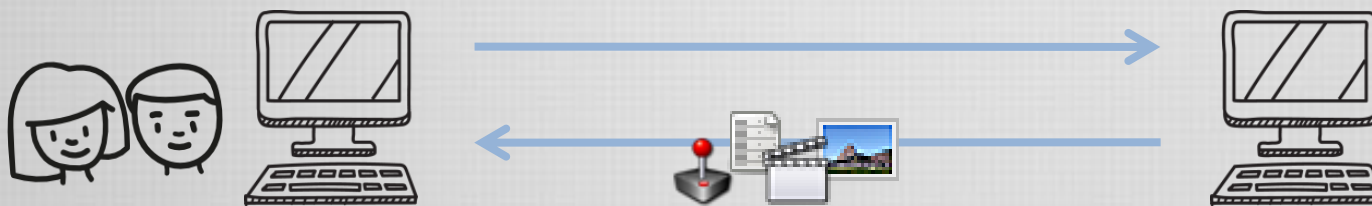
Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards

Sin embargo, se entiende que no necesariamente debe haber SOAP de por medio.

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

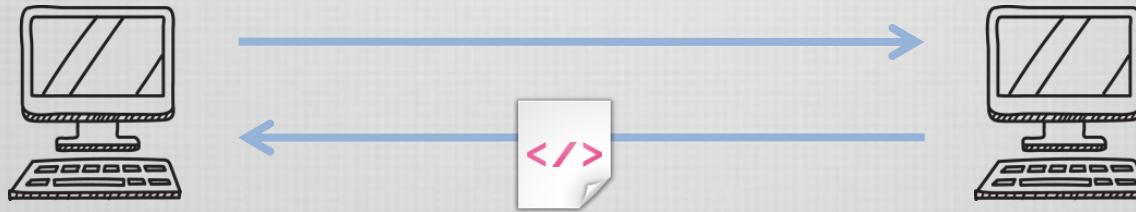
No tiene dependencias de ningún sistema operativo o lenguaje de programación.

La web para los humanos



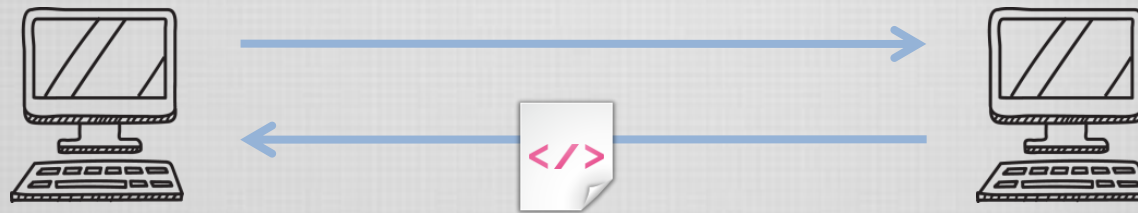
La web es en realidad una **red de servicios centrados en el humano**.
El consumidor final de los recursos de la web es el usuario humano.

La web para las aplicaciones



*Los servicios web son la **web centrada en las aplicaciones**.
Los consumidores del servicio no son humanos sino programas, sin
importar la plataforma ni la tecnología nativa.*

Web Service – la web para los humanos



XML
(*extensible Markup Language*)

estructurar información general,
con independencia de la plataforma
fácil de procesar
legible por los humanos

DTD

XML Schema

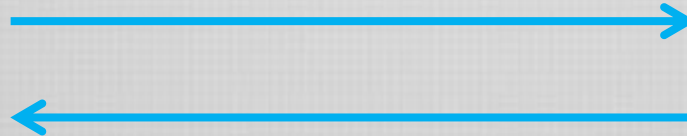
XPath

XSLT

Web Service – XML-RPC

XML-RPC es simplemente la invocación remota de funciones via web.

```
<?xml version="1.0"?>
<methodCall>
  <methodName>circleArea</methodName>
  <params>
    <param>
      <value><double>2.41</double></value>
    </param>
  </params>
</methodCall>
```

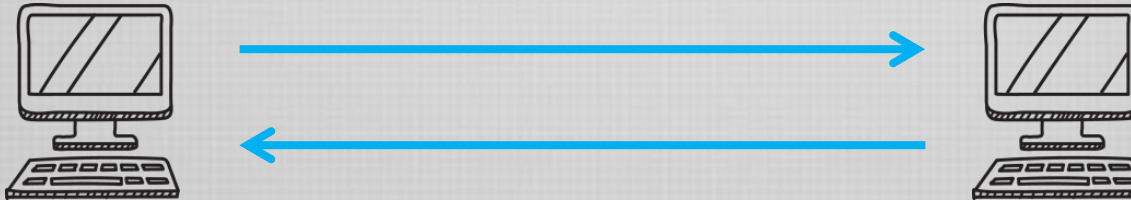


```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><double>18.24668429131</double></value>
    </param>
  </params>
</methodResponse>
```

Web Service – SOAP

SOAP es un mecanismo general para la interoperabilidad de sistemas.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:getTemp xmlns:ns1="urn:xmethods-Temperature"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      <zipcode xsi:type="xsd:string">10016</zipcode>
    </ns1:getTemp>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

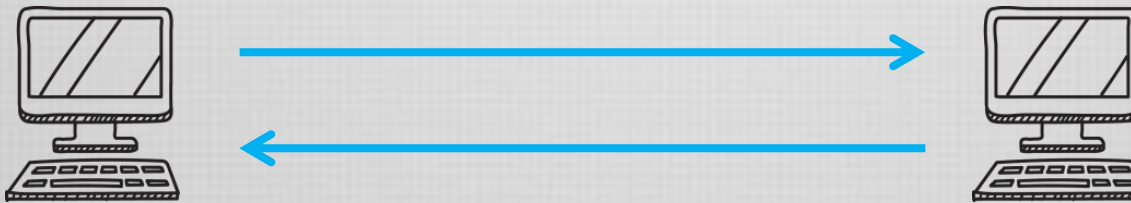


```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:getTempResponse xmlns:ns1="urn:xmethods-Temperature"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      <return xsi:type="xsd:float">71.0</return>
    </ns1:getTempResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Web Service – REST

REST es una metodología simple para la implementación de web services.

GET `http://miserviciosrest.com/alumnos/dcic/materia/rpa`



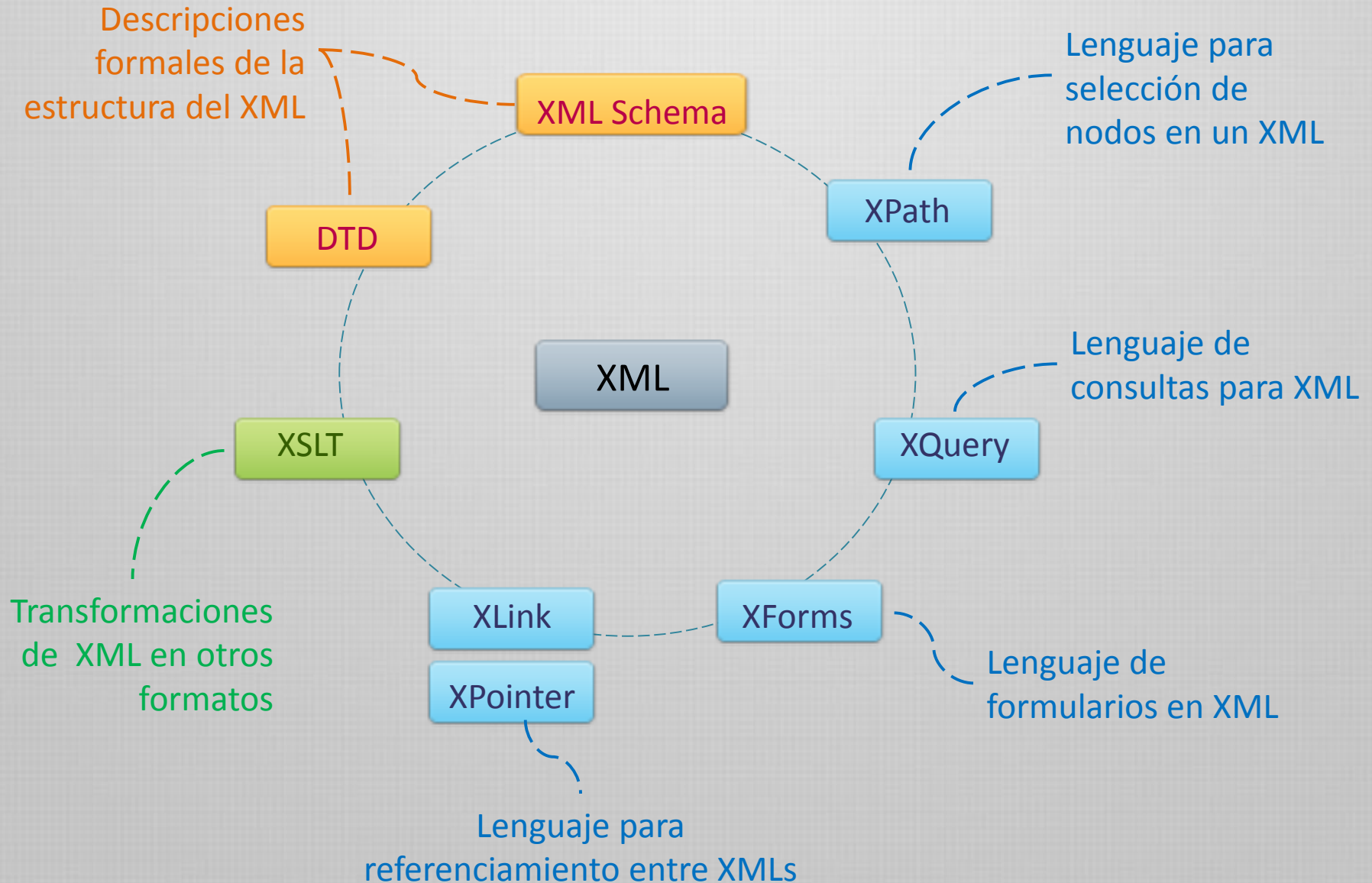
```
<?xml version='1.0' encoding='UTF-8'?>
<alumnos>
  <alumno>
    <nombre>Millhouse</nombre>
    <lu>12345</lu>
  </alumno>
  <alumno>
    <nombre>Bart</nombre>
    <lu>67890</lu>
  </alumno>
  <alumno>
    <nombre>Nelson</nombre>
    <lu>24680</lu>
  </alumno>
</alumnos>
```

Create
Retrieve
Update
Delete

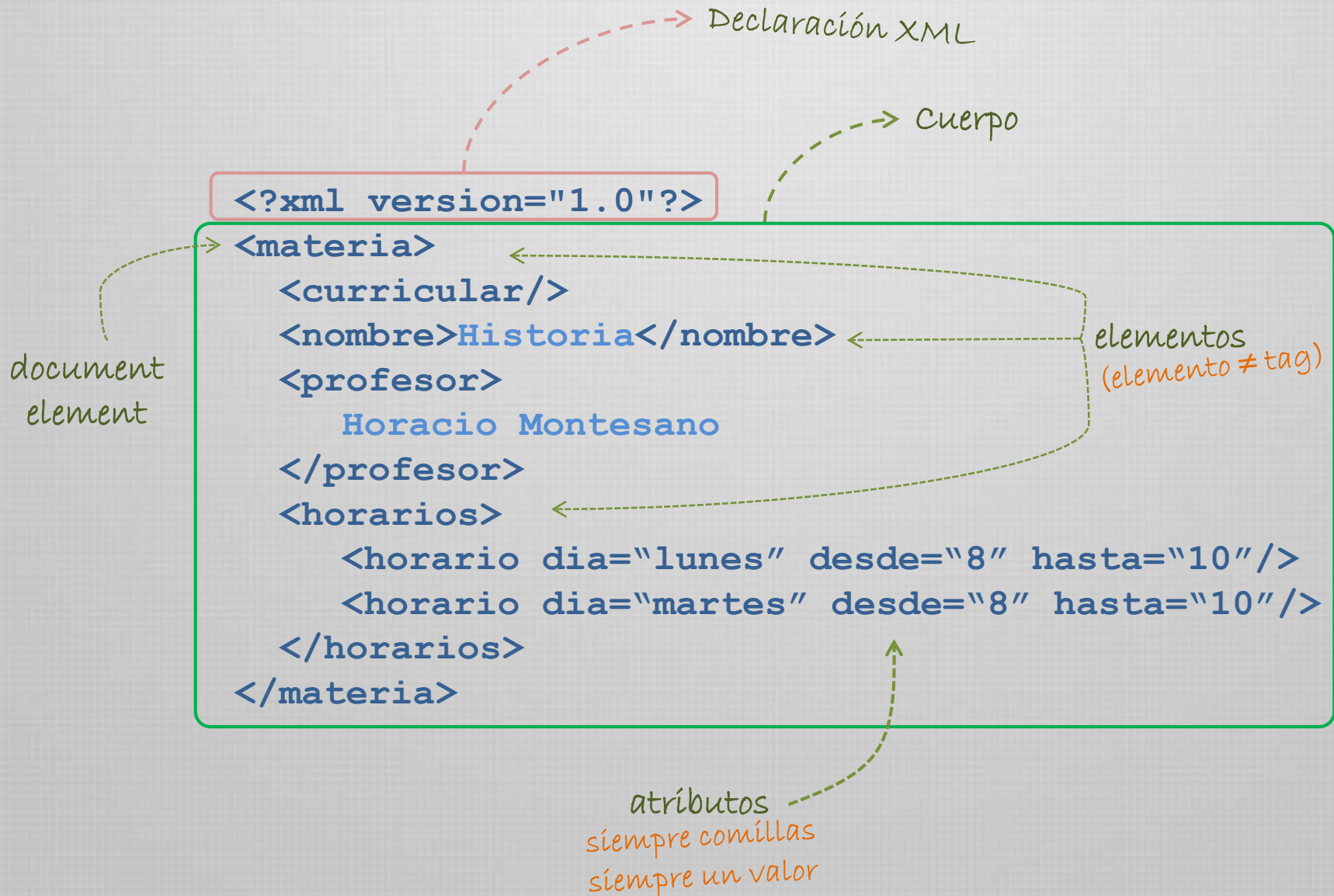
GET
POST
PUT
DELETE
HEAD

XML

XML – algunas tecnologías asociadas



XML - anatomía



XML - namespaces

<pre><producto> <nombre> Flatron L17 </nombre> <descripcion> Monitor LCD de 17 pulgadas. </descripcion> </producto></pre>		<pre><contacto> <nombre> Juan Perugia </nombre> <telefono tipo="cel">12345678</telefono> <telefono tipo="oficina">42346</telefono> </contacto></pre>
---	--	--

Un mismo elemento puede significar cosas diferentes dependiendo del contexto.

Para evitar *colisiones de tags* —————→ *namespaces*
Colección de vocabularios

Se define un nombre de prefijo, identificado por un URI.

xmlns:prefix="URI"

XML - namespaces

```
<proveedor>
  <producto>
    <nombre>...</nombre>
    ...
  </producto>
  <contacto>
    <nombre>...</nombre>
    ...
  </contacto>
</proveedor>
```

```
<proveedor xmlns:cont="http://aa.com/contactos">
  <producto>
    <nombre>...</nombre>
    ...
  </producto>
  <contacto>
    <cont:nombre>...</cont:nombre>
    ...
  </contacto>
</proveedor>
```

El espacio de nombres puede aplicarse a atributos también:

```
<untag cont:unatr="valor" >
```

XML – tipos de documentos

Si hay libertad de definir tags

¿cómo sabemos cuál es la estructura correcta de un XML?

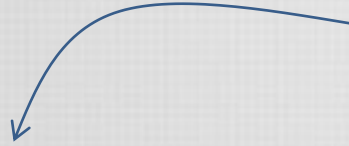
```
<telefono tipo="cel">12345678</telefono>  
<telefono tipo="cell">12345678</telefono>  
<telefono tipo="celular">12345678</telefono>
```

¿cuáles son los elementos válidos?

¿cuáles elementos pueden estar anidados?

¿existe algún orden determinado para los elementos anidados?

XML - validez



bien formado

si sigue las reglas del estándar XML.

(tags cerrados, uso correcto de comillas, anidación correcta, etc)

válido

si cumple un conjunto de reglas estructurales, especificadas en
DTD – Document Type Definition, o
XSD – XML Schema Definitions.

*Estos documentos dicen cómo es la **estructura correcta** del documento XML.
Define el tipo de dato.*

Un documento XML puede ser bien formado y no válido.
Un documento XML que no está bien formado no puede ser válido.



La validación es esencial antes del procesamiento,
especialmente cuando la fuente es externa

DTD- Document Type Definition

La definición del DTD incluye declaraciones para elementos y atributos del XML.
(la misma sintaxis, ya sea un DTD internal o external)

`<!ELEMENT elemento specif_contenido>`



Mixed -----> Caracteres (**#PCDATA**) y otros elementos

Children -----> Sólo otros elementos (sin *text-nodes*)

Wildcards	?	La expresión es opcional
	exp1 exp2	Secuencia de expresiones
		Alternativa de expresiones
	exp1 - exp2	La primera expresión, pero no la segunda
	+	Una o más ocurrencias de la expresión
	*	Cero o más ocurrencias de la expresión

DTD- Document Type Definition - elementos

```
<!ELEMENT apellido (#PCDATA)>
```

```
<!ELEMENT contacto (nombre,telefono,direccion)>
```

```
<!ELEMENT vehiculo (patente|numidentificacion)>
```

```
<!ELEMENT img EMPTY>
```

```
<!ELEMENT descripcion ANY>
```

```
<!ELEMENT subseccion (#PCDATA)>
```

```
<!ELEMENT seccion (#PCDATA|subseccion)>
```

```
<!ELEMENT articulo (titulo?, (parrafo+, grafico)*)>
```


DTD- Document Type Definition - atributos

`<!ATTLIST elemento atributo tipo defaultdecl>`



`<!ATTLIST auto marca CDATA #REQUIRED>`

`<auto marca='Chevrolet'>` ✓
`<auto marca=''>` ✓
`<auto>` ✗

`<!ATTLIST auto marca CDATA #IMPLIED>`

`<auto marca='Chevrolet'>` ✓
`<auto marca=''>` ✓
`<auto>` ✓

`<!ATTLIST auto modelo CDATA #FIXED "98">`

`<auto marca='Chevrolet'>` ✓
`<auto modelo='98'>` ✓
`<auto modelo='2012'>` ✗

DTD- Document Type Definition - atributos

```
<!ATTLIST contacto telefono (cel|oficina|casa)>
```

```
<!ATTLIST mensaje prioridad (urgente|normal) "normal">
```

```
<!ATTLIST producto oferta (si|no) #REQUIRED>
```

```
<!ATTLIST mensaje firma CDATA #IMPLIED>
```

```
<!ATTLIST curso nombre NMTOKEN>
```

```
<!ATTLIST curso correlativas NMTOKENS "ninguna">
```

DTD es simple, pero...

No es posible indicar, por ejemplo, que algún atributo debe ser un número positivo.

No es posible condicionar la existencia de atributos según otros atributos.

No es posible indicar el orden de los elementos

Defaults para atributos, no para elementos

XML Schemas

XML Schema es un estándar para la especificación de la estructura de un XML.

Más expresivo que XML

Escrito en XML

Todo esquema XML comienza con el elemento **schema**.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
    <!-- contenido -->  
</xsd:schema>
```

Los elementos se declaran con el elemento **element**.

```
<xsd:element name="nombre_elemento" type="tipo_elemento"/>
```

XML Schemas – simple types

```
<xs:element name="autor" type="xs:string"/>
<xs:element name="precio" type="xs:decimal"/>
<xs:element name="edad">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xsd:simpleType name="tipoPublicacion">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="Book"/>
    <xsd:enumeration value="Magazine"/>
    <xsd:enumeration value="Journal"/>
    <xsd:enumeration value="Online"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="pubType" type="tipoPublicacion"/>
```

XML Schemas – simple types

```
<xs:element name="jeans_size">
  <xs:simpleType>
    <xs:union memberTypes="sizebyno sizebystring" />
  </xs:simpleType>
</xs:element>
```

```
<xs:simpleType name="sizebyno">
  <xs:restriction base="xs:positiveInteger">
    <xs:maxInclusive value="42"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="sizebystring">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>
```


XSD – Ejemplos

```
<xs:element name="listaEnteros" type="tipoListaEnteros">
```

```
<xs:simpleType name="tipoListaEnteros">
```

```
<xs:list itemType="xs:integer"/>
```

```
</xs:simpleType>
```

Ejemplo válido según la regla anterior:

```
<listaEnteros>343 1231 9 7654</listaEnteros>
```


XSD – XML Schema Definitions - tipos complejos

Los tipos complejos son agregados de tipos simples

xsd:sequence	Secuencia ordenada de partes
xsd:choice	Selección entre opciones
xsd:all	Todas las partes en cualquier orden

```
<xs:complexType name="tipoDireccion">
  <xs:sequence>
    <xs:element name="calle" type="xs:string"/>
    <xs:element name="ciudad" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="direccion" type="tipoDireccion">
```

Ejemplo válido según la regla anterior:

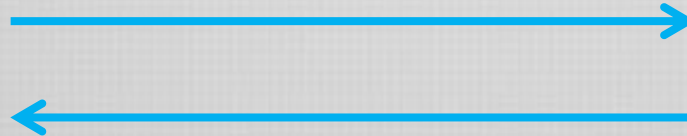
```
<direccion>
  <calle>Fake Street</calle>
  <ciudad>Ciudad Capital</ciudad>
</direccion>
```

XML-RPC

Web Service – XML-RPC

XML-RPC es simplemente la invocación remota de funciones via web.

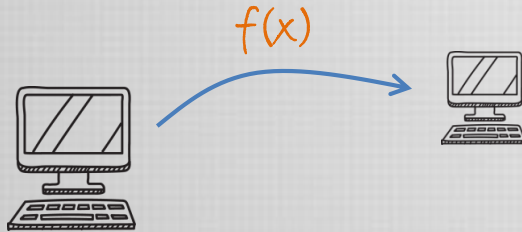
```
<?xml version="1.0"?>
<methodCall>
  <methodName>circleArea</methodName>
  <params>
    <param>
      <value><double>2.41</double></value>
    </param>
  </params>
</methodCall>
```



```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><double>18.24668429131</double></value>
    </param>
  </params>
</methodResponse>
```

Web Service – XML-RPC

XML-RPC = XML-based *Remote Procedure Call*



La función solicitada se ejecuta en otra aplicación, incluso en otra máquina separada.

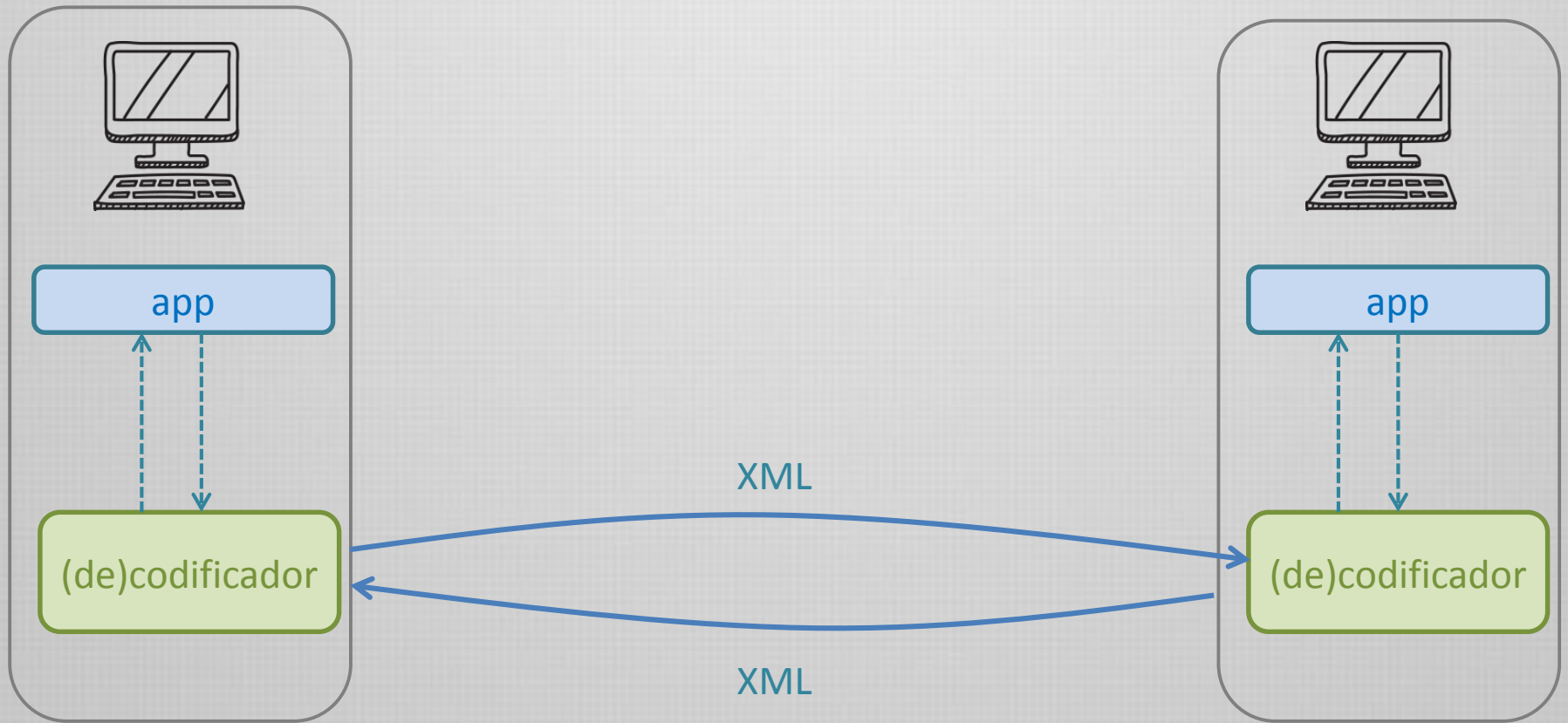
Para que la comunicación entre partes heteróneas pueda realizarse, la llamada y los datos deben formatearse apropiadamente (*marshalling*).

XML-RPC define formatos específicos para esta comunicación

XML-RPC transporta requerimientos al servidor por medio de mensajes HTTP POST

XML-RPC es tal vez la forma más primitiva de implementar servicios web.

Web Service – XML-RPC



Los mensajes de request se transmiten via HTTP utilizando el método POST

Recordemos ...



POST /empleados/despedir.cgi HTTP/1.0

From: burns@snppl.com

User-Agent: HTTPTool/1.0

Content-Type: application/x-www-form-urlencoded

Content-Length: 23

nombre=Homero§or=7G

cuerpo del mensaje

XML-RPC - elementos

XML-RPC requiere que los datos pasados entre las partes estén en un elemento XML denominado *value*

`<value> </value>`

El contenido identifica el tipo de dato

`<value><int>5</int></value>`

32bits

`<value><i4>5</i4></value>`

intercambiables

`<value><double>2.34334</double></value>`

`<value><boolean>0</boolean></value>`

omitible

`<value><string>Hola!</string></value>`

`<value><dateTime.iso8601>`

`20141113T18:35:00`

`</dateTime.iso8601></value>`

`<value><base64>Hola!</base64></value>`

XML-RPC - elementos

Pueden especificarse también tipos de datos estructurados

Arreglos en XML-RPC:

```
<value>
  <array>
    <data>
      <value>...</value>
      <value>...</value>
      ...
    </data>
  </array>
</value>
```

*cualquier tipo de valores,
inclusivamente otros arreglos*

Los arreglos de este tipo son todos de índice numérico

```
$a = array("Hola", "Mundo");
```

```
<value>
  <array>
    <data>
      <value><string>Hola</string></value>
      <value><string>Mundo</string></value>
    </data>
  </array>
</value>
```

XML-RPC - elementos

Estructuras más complejas (como arreglos asociativos) se representan en XML-RPC con el tipo de dato *structure*, una colección de valores nombrados.

```
<value>
  <struct>
    <member>
      <name>..</name>
      <value>..</value>
    </member>
    ...
  </struct>
</value>
```

*Puede haber varios elementos
de tipo member*

```
$a = array(
  "nombre"=>"Juan",
  "deuda"=>"23.5"
);
```

```
<value>
  <struct>
    <member>
      <name>nombre</name>
      <value>
        <string>Juan</string>
      </value>
    </member>
    <member>
      <name>deuda</name>
      <value>
        <double>23.5</double>
      </value>
    </member>
  </struct>
</value>
```

XML-RPC Requests

Los requerimientos y respuestas también tienen un formato XML predefinido.

Estructura del XML-RPC Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>nombre del método</methodName>
  <params>
    <param>
      parametro1
    </param>
    <param>
      parametro2
    </param>
    ...
  </params>
</methodCall>
```

A-Z, a-z, 0-9
· : _ /

Puede no haber parámetros y
se omite el elemento params

XML-RPC Requests

```
getPelículas( año estreno, nombre del actor, incluir descripcion );
```

```
getPelículas(80,"Robert De Niro",true);
```

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>getPelículas</methodName>
  <params>
    <param>
      <value><int>80</int></value>
    </param>
    <param>
      <value><string>Robert de Niro</string></value>
    </param>
    <param>
      <value><boolean>1</boolean></value>
    </param>
  </params>
</methodCall>
```

El orden de los parámetros es importante.

El tipo de los parámetros es importante si lo es para el receptor.

PHP, JavaScript —> *loosely typed*

XML-RPC Requests

XML-RPC requiere headers HTTP en cada *request*.

```
POST /myservice.php HTTP/1.0
User-Agent: PHP XMLRPC 1.0
Host: xmlrpc.usefulinc.com
Content-Type: text/xml
Content-Length: 216
```

```
<data></data>
```

Implementación
XML-RPC
(el menos observado)

Server que
atenderá el pedido

Constante
(excepto en JSON)

XML-RPC Requests

```
POST /rpchandler HTTP/1.0
User-Agent: AcmeXMLRPC/1.0
Host: xmlrpc.example.com
Content-Type: text/xml
Content-Length: 165
```

```
<?xml version="1.0"?>
<methodCall>
  <methodName>getCapitalCity</methodName>
  <params>
    <param>
      <value>
        <string>England</string>
      </value>
    </param>
  </params>
</methodCall>
```


XML-RPC Response

El formato de una respuesta XML-RPC también es fijo y predefinido

Estructura del XML-RPC response:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value> valor del resultado </value>
    </param>
  </params>
</methodResponse>
```

```
<?xml version="1.0"?>
<methodResponse>
  <params/>
</methodResponse>
```



void functions

XML-RPC Response

Estructura del mensaje de error XML-RPC

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>55</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>Mensaje Error</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

*No existen guías ni
códigos estandarizados*

XML-RPC Response

 *Siempre!*
HTTP/1.1 200 OK

Date: Sun, 29 Apr 2001 12:08:58 GMT

Server: Apache/1.3.12 (Unix) Debian/GNU PHP/4.0.2

Connection: close

Content-Type: text/xml

Content-length: 133

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

```
<methodResponse>
```

```
<params>
```

```
<param>
```

```
<value><string>Michigan</string></value>
```

```
</param>
```

```
</params>
```

```
</methodResponse>
```

JSON-RPC

Protocolo RPC basado en JSON

—————→ *un mensaje, un objeto JSON*

request

f(x)



- jsonrpc:** versión del protocolo. Debe ser "2.0".
- method:** nombre del método a invocar (no usar rpc como prefijo)
- params:** estructura con los argumentos del método.
- id:** identificador del *cliente* (*String*, *Number*, *NULL*, si se incluye)

Los parámetros deben ser provistos como una estructura JSON

por posición —→ un **array**, con el **orden** esperado.

por nombre —→ un **objeto**, con los mismos **nombres** esperados.

JSON-RPC

Protocolo RPC basado en JSON

→ *un mensaje, un objeto JSON*

request

$f(x)$



- jsonrpc:** versión del protocolo. Debe ser "2.0".
- method:** nombre del método a invocar (no usar rpc como prefijo)
- params:** estructura con los argumentos del método.
- id:** identificador del *cliente* (*String*, *Number*, *NULL*, si se incluye)

```
{  
  "jsonrpc": "2.0",  
  "method": "sumar",  
  "params": [42, 23],  
  "id": 1  
}
```

```
{  
  "jsonrpc": "2.0",  
  "method": "getSueldo",  
  "params":  
    { "legajo": 12345,  
      "mes": 5 },  
  "id": 3  
}
```

JSON-RPC

Protocolo RPC basado en JSON

—————→ *un mensaje, un objeto JSON*

request

$f(x)$



- jsonrpc:** versión del protocolo. Debe ser "2.0".
- method:** nombre del método a invocar (no usar rpc como prefijo)
- params:** estructura con los argumentos del método.
- id:** identificador del *cliente* (*String*, *Number*, *NULL*, si se incluye)

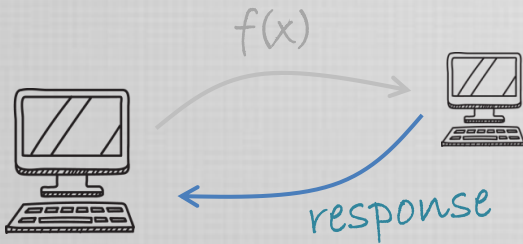
Una **notificación** es un objeto request sin "id"
El servidor no debe responder a las notificaciones.

```
{  
  "jsonrpc": "2.0",  
  "method": "update",  
  "params": [1,2,3,4,5]  
}
```

JSON-RPC

Protocolo RPC basado en JSON

—————→ *un mensaje, un objeto JSON*



- jsonrpc:** versión del protocolo. Debe ser "2.0".
- result:** requerido si no hay error. El valor lo determina el método del servidor.
- error:** requerido si hubo error. El valor debe ser un objeto predefinido.
- id:** requerido. Debe ser el mismo valor que el id del request.

```
{  
  "jsonrpc": "2.0",  
  "method": "sumar",  
  "params": [42, 23],  
  "id": 65  
}
```

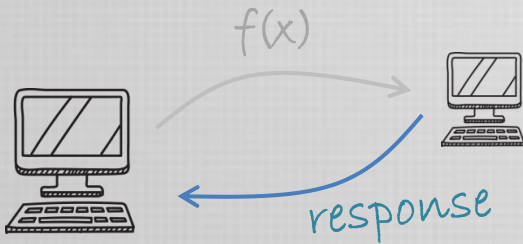


```
{  
  "jsonrpc": "2.0",  
  "result": 65,  
  "id": 65  
}
```


JSON-RPC

Protocolo RPC basado en JSON

—————→ *un mensaje, un objeto JSON*



- jsonrpc:** versión del protocolo. Debe ser "2.0".
- result:** requerido si no hay error. El valor lo determina el método del servidor.
- error:** requerido si hubo error. El valor debe ser un objeto predefinido.
- id:** requerido. Debe ser el mismo valor que el id del request.

- error** {
- code:** número entero.
 - message:** string. descripción del error
 - data:** opcional. Información adicional del error.

```
{
  "jsonrpc": "2.0",
  "error": {"code": -32601,
            "message": "Method not found"},
  "id": "1"
}
```

JSON-RPC - batch

```
[
  { "jsonrpc": "2.0",
    "method": "suma",
    "params": [1,2,4],
    "id": "1"
  },
  { "jsonrpc": "2.0",
    "method": "saludo",
    "params": [7]
  },
  { "jsonrpc": "2.0",
    "method": "resta",
    "params": [8,2],
    "id": "2"
  },
  { "foo": "boo"
  },
  { "jsonrpc": "2.0",
    "method": "ordinaldia",
    "params": {"dia": "lunes"},
    "id": "5"
  }
]
```

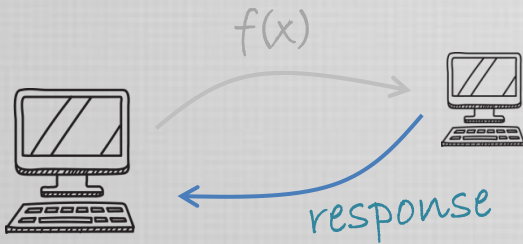


```
[
  { "jsonrpc": "2.0",
    "result": 7,
    "id": "1"
  },
  { "jsonrpc": "2.0",
    "result": 6,
    "id": "2"
  },
  { "jsonrpc": "2.0",
    "error": {
      "code": -32600,
      "message": "Invalid Request"
    },
    "id": null
  },
  { "jsonrpc": "2.0",
    "result": 2,
    "id": "5"
  }
]
```

JSON-RPC

Protocolo RPC basado en JSON

—————→ *un mensaje, un objeto JSON*



- jsonrpc:** versión del protocolo. Debe ser "2.0".
- result:** requerido si no hay error. El valor lo determina el método del servidor.
- error:** requerido si hubo error. El valor debe ser un objeto predefinido.
- id:** requerido. Debe ser el mismo valor que el id del request.

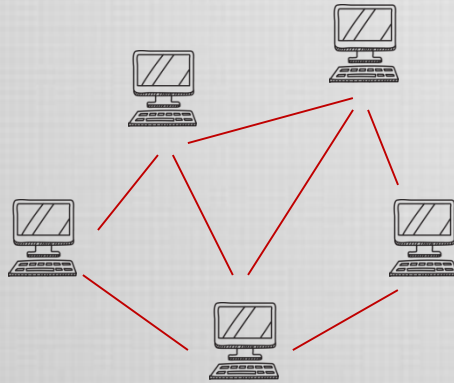
Códigos de errores

-32700	Parse error	<i>Invalid JSON was received by the server. An error occurred on the server while parsing the JSON text.</i>
-32600	Invalid Request	<i>The JSON sent is not a valid Request object.</i>
-32601	Method not found	<i>The method does not exist / is not available.</i>
-32602	Invalid params	<i>Invalid method parameter(s).</i>
-32603	Internal error	<i>Internal JSON-RPC error.</i>
-32000 to -32099	Server error	<i>Reserved for implementation-defined server-errors</i>

SOAP

SOAP

SOAP → *Simple Object Access Protocol*

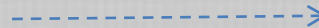


intercambio de
información estructurada y tipada
entre puntos de un
ambiente distribuído descentralizado

Es un lenguaje que define el
formato de mensajes



independiente de la plataforma
basado en XML
con facilidades de extensión



*intercomunicación de
aplicaciones*

SOAP

1998

Microsoft, IBM, Lotus (y otras)

Primera versión, enfocada en definir un sistema de tipos



1999

Primera especificación - nombre SOAP.

basado en *XML Schema Type System* + *headers de protocolos*
"vendor wars"



2003

SOAP - W3C Recommendation



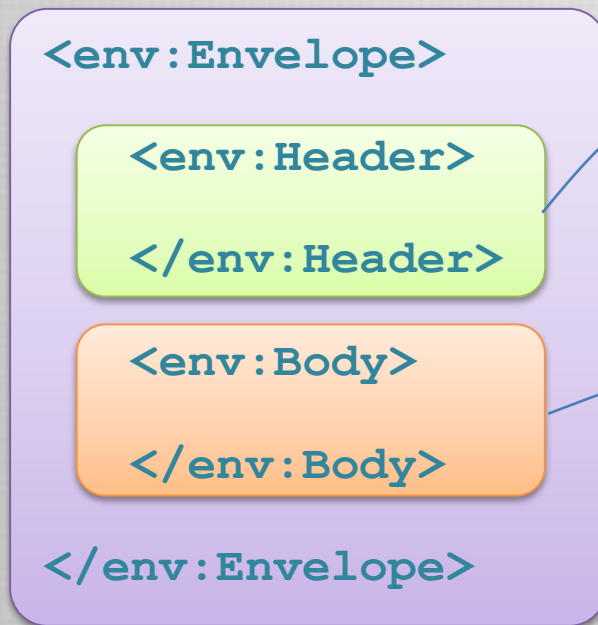
2007

SOAP 1.2 - Messaging Framework

No es más un acrónimo.

SOAP - estructura

La estructura general de un mensaje SOAP es simple.
Se lo denomina **SOAP Envelope** y es un documento XML sin DTD ni PI



Información de control, formada por elementos denominados *header blocks*. (descripciones del mensaje, instrucciones de procesamiento, etc)

Es opcional

Transporta la información que es relevante para los extremos de la comunicación.

Es obligatorio

SOAP en HTTP

SOAP es lo suficientemente general como para usar otros protocolos de Internet.

En el contexto de los servicios web, los mensajes SOAP serán enviados via HTTP.

El mensaje HTTP apropiado es POST.

```
POST /soap HTTP/1.1
Host: localhost
Connection: Keep-Alive
User-Agent: PHP-SOAP/5.3.1
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 471
```

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ns1="http://example.localhost/index/soap"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://www.w3.org/2003/05/soap-encoding">
  <env:Body>
    <ns1:getLibro env:encodingStyle="http://www..soap-encoding">
      <param0 xsi:type="xsd:string">1456594788</param0>
    </ns1:getLibro>
  </env:Body>
</env:Envelope>
```

getLibro(1456594788)

SOAP - ejemplo

```
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
```

```
<s:Header>
```

```
<m:transaction xmlns:m="soap-transaction" s:mustUnderstand="true">
```

```
<transactionID>1234</transactionID>
```

```
</m:transaction>
```

```
</s:Header>
```

```
<s:Body>
```

```
<n:purchaseOrder xmlns:n="urn:OrderService">
```

```
<from>
```

```
<person>Juan Perugia</person>
```

```
</from>
```

```
<to>
```

```
<person>Steven Spielberg</person>
```

```
</to>
```

```
<order>
```

```
<quantity>1</quantity>
```

```
<item>Casette Video</item>
```

```
</order>
```

```
</n:purchaseOrder>
```

```
</s:Body>
```

```
</s:Envelope>
```


SOAP - ejemplo

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>Åke Jógvan Øyvind</n:name>
    </n:passenger>
  </env:Header>
  <env:Body>
    <p:itinerary xmlns:p="http://travel.example.org/reservation/travel">
      <p:departure>
        <p:departing>New York</p:departing>
        <p:arriving>Los Angeles</p:arriving>
      </p:departure>
      <p:return>
        <p:departing>Los Angeles</p:departing>
        <p:arriving>New York</p:arriving>
      </p:return>
    </p:itinerary>
  </env:Body>
</env:Envelope>
```


SOAP – ejemplo - fault

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.../envelope/">
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
  <faultcode>ns1:DBError</faultcode>
  <faultstring>A database error has occurred</faultstring>
  <detail>
    <ns1:DBUnavailableFault>
      <DBMessage>Unable to connect to database</DBMessage>
      <RetryInMinutes>60</RetryInMinutes>
    </ns1:DBUnavailableFault>
  </detail>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

WSDL

WSDL es una gramática en XML utilizada para describir servicios web
Significa Web Services Description Language.

No es estrictamente necesario utilizar WSDL, aunque es lo recomendable.

El documento WSDL debe definir
el servicio completo,
como una colección de operaciones
que reciben y/o retornan datos estructurados.

Esta descripción es *abstracta*

Naturalmente, es necesario describir la *forma de comunicación*.

El documento WSDL debe definir, además,
qué tipo de protocolo de transporte se utilizará y
cuál es el formato de los mensajes que se intercambiarán.

Esta la parte *concreta* de la descripción del servicio

El nivel de abstracción buscado y la modularidad en las descripciones hace que los documentos WSDL sean algo extensos y confusos.

WSDL – partes

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <!-- definiciones de tipos utilizados -->
  </types>

  <message>
    <!-- definicion abstracta de los datos que se transmiten -->
  </message>

  <portType>
    <!-- operaciones abstractas + input y output de mensajes -->
  </portType>

  <binding>
    <!-- protocolo concreto y formato de datos especifico -->
  </binding>

  <service>
    <!-- ubicaciones y bindings del servicio -->
  </service>

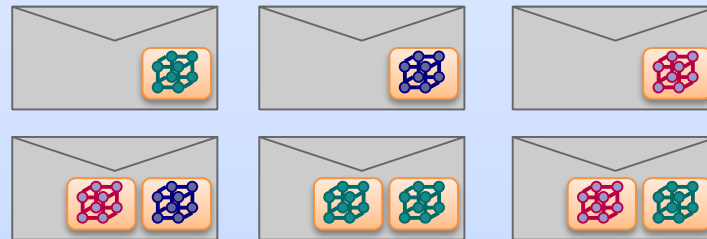
</definitions>
```

WSDL – definiciones *abstractas*

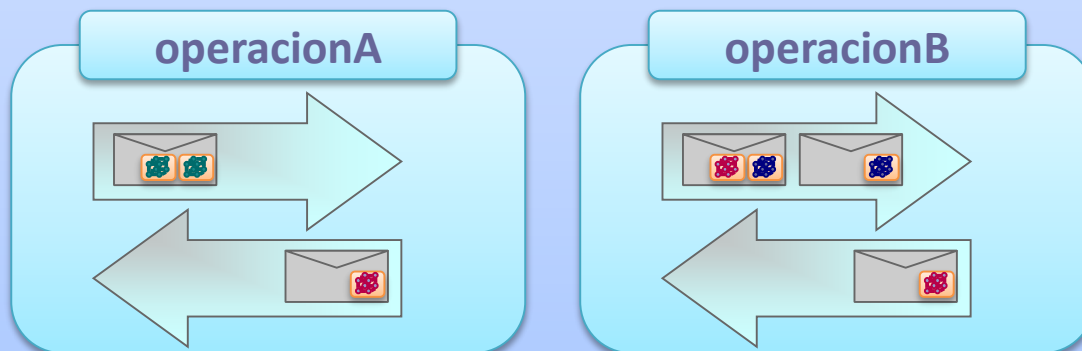
<types>



<message>



<portType>



WSDL – definiciones *abstractas* - tipos

```
<?xml version="1.0"?>
<definitions name="StockQuote" ...namespaces...>

  <types>
    <schema ...namespaces xsd...>

      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>

      <element name="TradePrice">
        <complexType>
          <all>
            <element name="price" type="float"/>
          </all>
        </complexType>
      </element>

    </schema>
  </types>
```


WSDL – definiciones *abstractas* – *mensajes + portTypes*

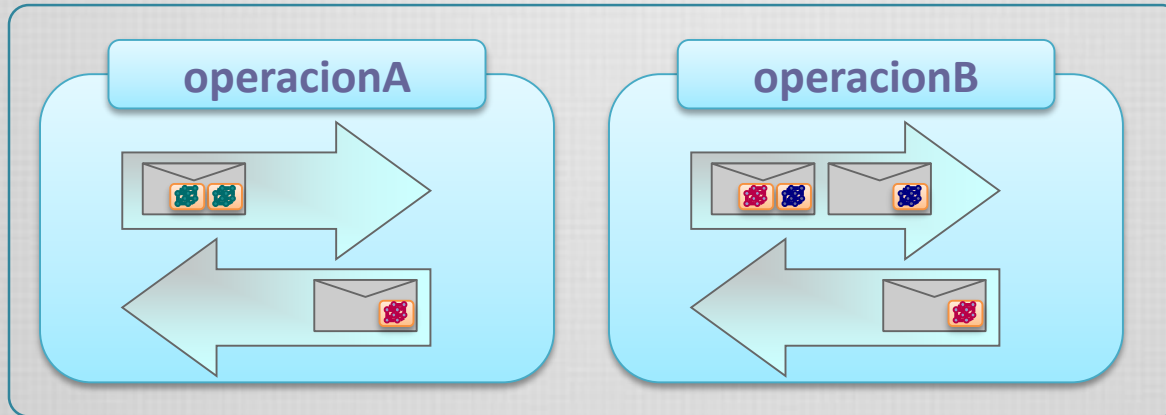
```
<message name="GetLastTradePriceInput">  
  <part name="body" element="xsd1:TradePriceRequest"/>  
</message>
```

```
<message name="GetLastTradePriceOutput">  
  <part name="body" element="xsd1:TradePrice"/>  
</message>
```

```
<portType name="StockQuotePortType">  
  <operation name="GetLastTradePrice">  
    <input message="tns:GetLastTradePriceInput"/>  
    <output message="tns:GetLastTradePriceOutput"/>  
  </operation>  
</portType>
```

WSDL – definiciones concretas – *bindings*

portType



binding

Servicio provisto

Transporte y formato



Una vez elegido el formato, debe indicarse cómo se formatea cada operación y sus mensajes.
binding portType - bindings operaciones - bindings mensajes

WSDL – definiciones concretas – *bindings y service*

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">

  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />

  <operation name="GetLastTradePrice">
    <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>

</binding>

<service name="StockQuoteService">
  <documentation>Mi servicio web SOAP</documentation>
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">
    <soap:address location="http://example.com/stockquote"/>
  </port>
</service>
```

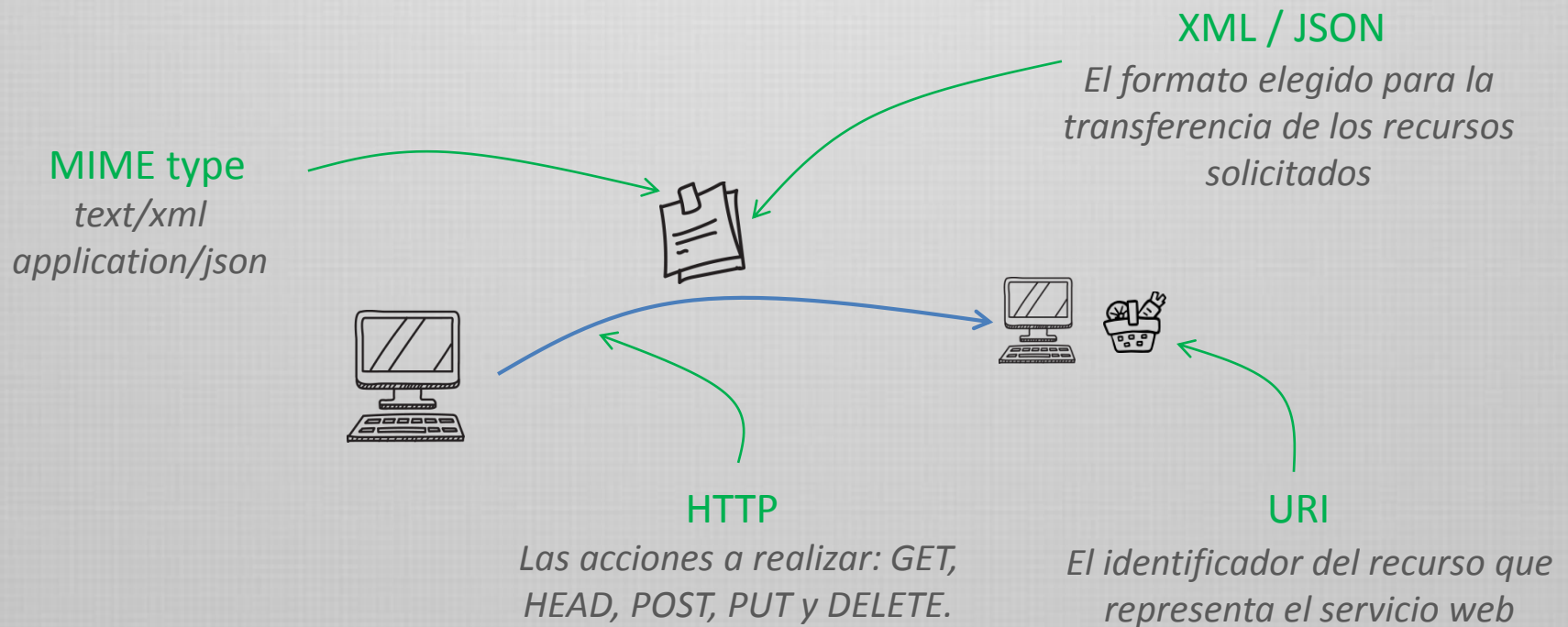
REST

REST

REST significa **R**epresentational **S**tate **T**ransfer.

No es un estándar como los anteriores, sino una abstracción de elementos de una arquitectura dentro de un sistema hipermedial distribuido.

Básicamente, traslada la configuración de un *request*, a las tecnologías ya provistas por la web.

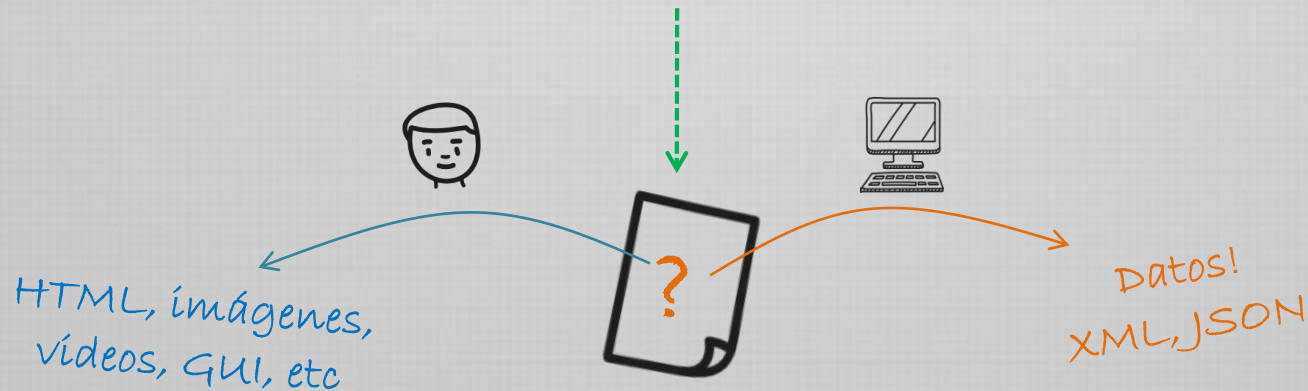


REST

REST ve el mundo como una colección de recursos. *abstracción de "datos"*

identificables por
medio de URI o URL

`http://unhost.com/ligafutbol/equipos/deportivoX`



REST – mensajes HTTP

REST asigna un rol especial a los mensajes HTTP, consistente con su uso web.

GET *Retrieve*

Recupera la representación de un recurso

HEAD *Retrieve*

Recupera metadatos de la representación de un recurso

POST *Create*

Estrictamente, crea un recurso.

Se usa también para actualizar un recurso o borrarlo

PUT *Update*

Actualiza un recurso.

Usualmente reemplazado por el método POST

DELETE *Delete*

Elimina un recurso.

Usualmente reemplazado por el método POST

<http://www.autopartes.com/partes>

```
<?xml version="1.0"?>
<p:partes xmlns:p="http://www...." xmlns:xlink="http://...">
  <p:autoparte id="ABC1" xlink:href="http://www.autopartes.com/partes/ABC1"/>
  <p:autoparte id="DEF2" xlink:href="http://www.autopartes.com/partes/DEF2"/>
  <p:autoparte id="GEW3" xlink:href="http://www.autopartes.com/partes/GEW3"/>
  <p:autoparte id="KLM1" xlink:href="http://www.autopartes.com/partes/KLM1"/>
</p:partes>
```

<http://www.autopartes.com/partes/DEF2>

```
<?xml version="1.0"?>
<p:partes xmlns:p="http://www...." xmlns:xlink="http://...">
  <p:autoparte>
    <p:id>DEF2</p:id>
    <p:desc>paragolpe delantero</p:desc>
    <p:auto>Fiat 600</p:auto>
    <p:stock>5</p:stock>
  </p:autoparte>
</p:partes>
```

REST – URLs

<http://maps.googleapis.com/maps/api/geocode/xml?address=Bahia Blanca>

```
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
  <status>OK</status>
  <result>
    <type>locality</type>
    <type>political</type>
    <formatted_address>
      Bahia Blanca, Buenos Aires, Argentina
    </formatted_address>
    <address_component>
      <long_name>Bahía Blanca</long_name>
      <short_name>Bahía Blanca</short_name>
      <type>administrative_area_level_2</type>
      <type>political</type>
    </address_component>
    ...
  </result>
</GeocodeResponse>
```

REST – URLs

<http://services.explorecalifornia.org/rest/tours.php?packageid=5>

```
<?xml version="1.0"?>
<tours>
  <tour>
    <tourId>14</tourId>
    <tourTitle>2 Days Adrift the Salton Sea</tourTitle>
    <packageId>5</packageId>
    <packageTitle>From Desert to Sea</packageTitle>
    <description>
      The Salton Sea is saltier than the Pacific,
      an unusual feat for inland body of water.
      And even though its salinity has risen over
      the years, due in part to lack ...
```


REST – URLs

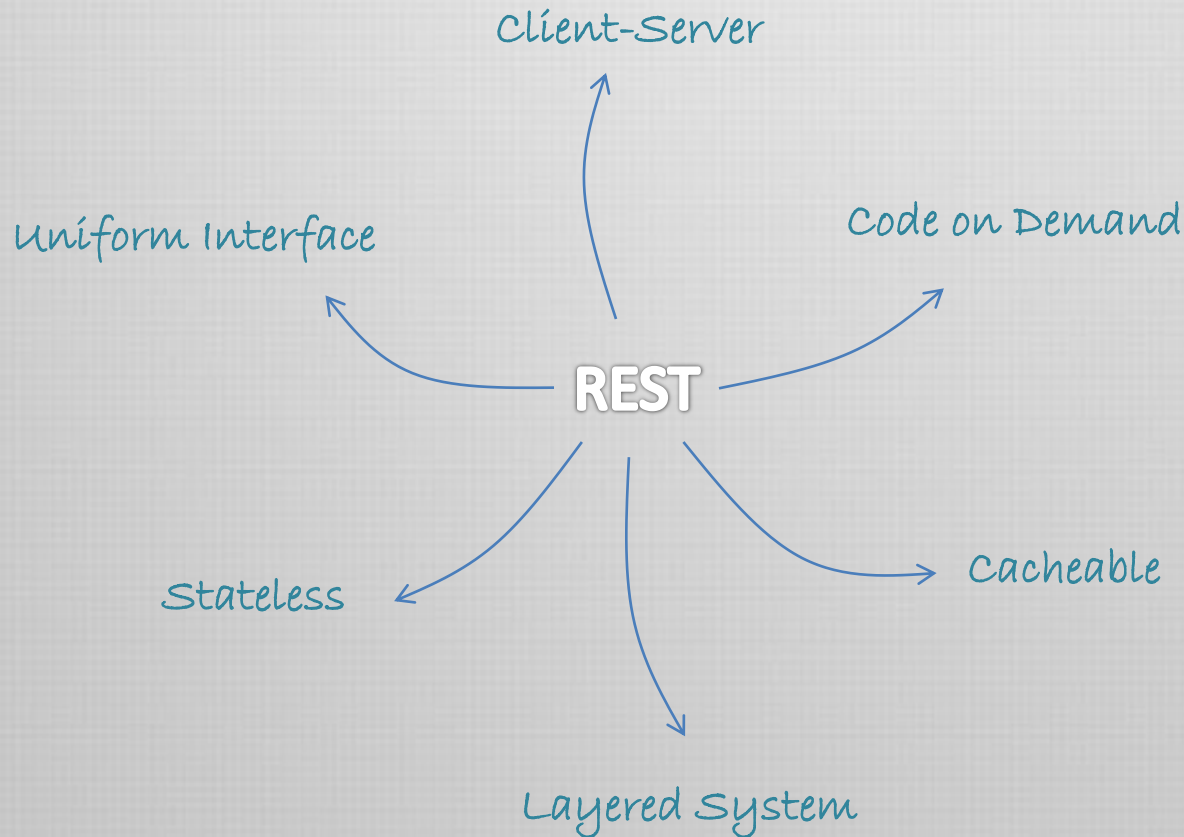
```
POST http://services.explorecalifornia.org/rest/tours.php HTTP/1.0
Host: services.explorecalifornia.org
Accept: text/xml
Content-Type: text/xml
Content-Length: 165
```

```
<?xml version="1.0"? Encoding="utf-8">
<parameters>
    <packageid>5<packageid>
</parameters>
```

REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

Roy Fielding



REST constraints

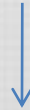
“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST

REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

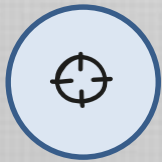
REST → Uniform Interface



principio de ingeniería: generalidad en la interfaz

simplifica la arquitectura

las implementaciones se desacoplan de los servicios que proveen



identificación de
recursos



manipulación de
recursos por su
representación



mensajes auto
descriptivos

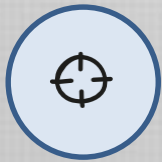


Hypermedia as
the Engine of
Application State

REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST → Uniform Interface



identificación de
recursos



manipulación de
recursos por su
representación



mensajes auto
descriptivos

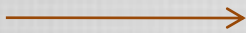


Hypermedia as
the Engine of
Application State

REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST



Uniform Interface



identificación de
recursos



manipulación de
recursos por su
representación



mensajes auto
descriptivos



Hypermedia as
the Engine of
Application State



*Los recursos son identificados por medio de un **URI**
Los recursos están conceptualmente separados de sus
representaciones.*

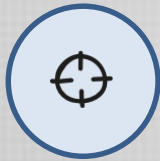
REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST



Uniform Interface



identificación de recursos



manipulación de recursos por su representación



mensajes auto descriptivos



Hypermedia as the Engine of Application State



Una representación = datos + metadatos

Tipo de datos = “media type”

La representación del recurso R puede variar en el tiempo $\longrightarrow M_R(t)$

La representación de un recurso puede ser el estado actual de un recurso o el estado deseado.

REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST



Uniform Interface



identificación de
recursos



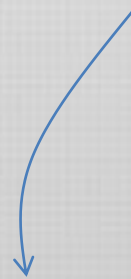
manipulación de
recursos por su
representación



mensajes auto
descriptivos



Hypermedia as
the Engine of
Application State



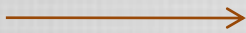
*Los mensajes deben contener toda la información necesaria
para procesar el mensaje.*

(Internet Media types, cache policies, etc)

REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST



Uniform Interface



identificación de
recursos



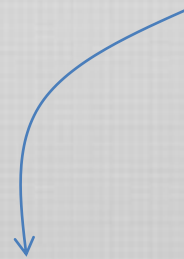
manipulación de
recursos por su
representación



mensajes auto
descriptivos



Hypermedia as
the Engine of
Application State



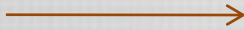
HATEOAS

*El modo de comunicación distribuida es **hipermedial**
Recursos desde y hacia los clientes.
Recursos linkeados (incluyen API)
Control de cache*

REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST



Stateless



Cada *request* del cliente debe contener toda la información
necesaria para comprender el *request*

No participa ningún contexto almacenado en el servidor

favorece **escalabilidad**

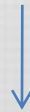
REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST



Cacheable



Los datos deben ser implícitamente o explícitamente identificados como **cacheables** o **no-cacheables**

reduce tráfico de red y favorece **performance**
favorece **escalabilidad**

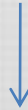
REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST



Client-server



“separation of concerns”

A los clientes no les incumbe aspectos de almacenamiento de datos

A los servidores no les incumbe el estado del usuario o su interfaz

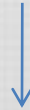
servidores y clientes reemplazables

favorece escalabilidad

REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST → Layered System



Arquitecturas en los extremos basadas en capas jerárquicas

Cada capa tiene su responsabilidad

Independencia de sustratos

reduce complejidad

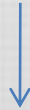
favorece escalabilidad

riesgo de overhead

REST constraints

“an architectural style consisting of the set of constraints applied to elements within the architecture”

REST → Code-on-demand



Los servidores puede extender la funcionalidad del cliente
El servidor transfiere lógica al cliente (scripts, applets, etc)

Es el único requerimiento opcional

simplifica la **complejidad** del cliente
clientes **extensibles**

Verbos HTTP

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

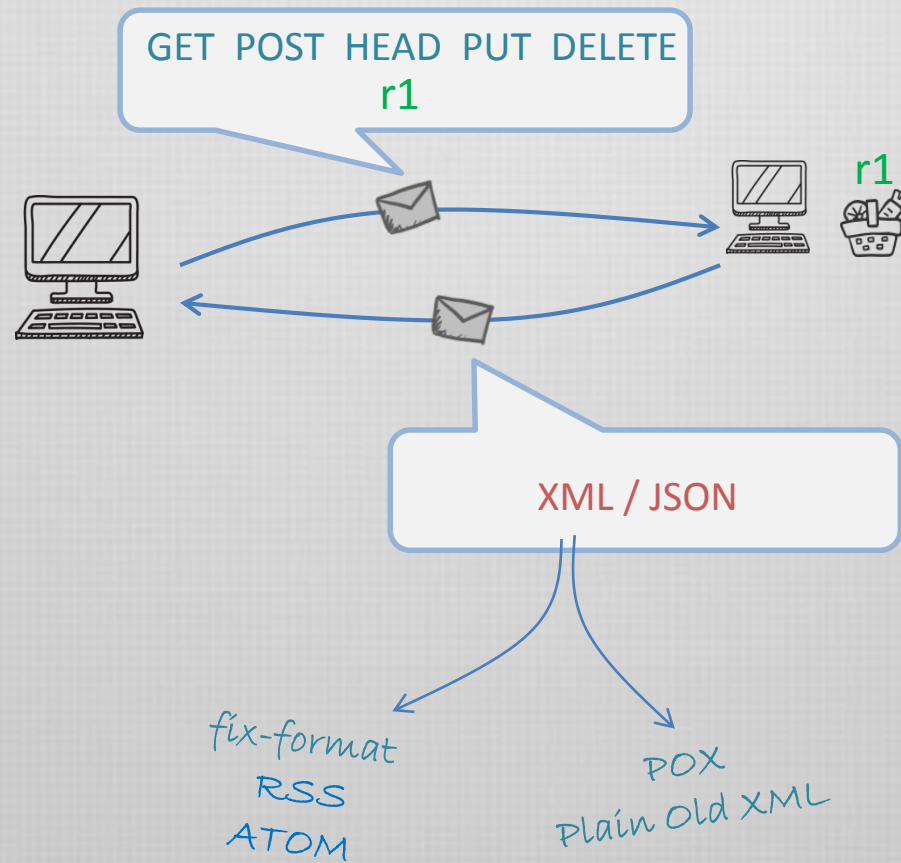
Uniform Interface

↪ **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)

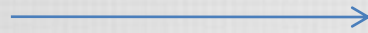
Verbo HTTP	/clientes	/clientes/{id}
GET	200 OK, lista de clientes.	200 OK, un cliente. 404 Not Found, id inválido o no encontrado
PUT	404 Not found, a menos que se desee cambiar la colección	200 OK o 204 No content. 404 Not Found, id inválido o no encontrado
POST	201 Created. Header Location con el URI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

Respuestas REST



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