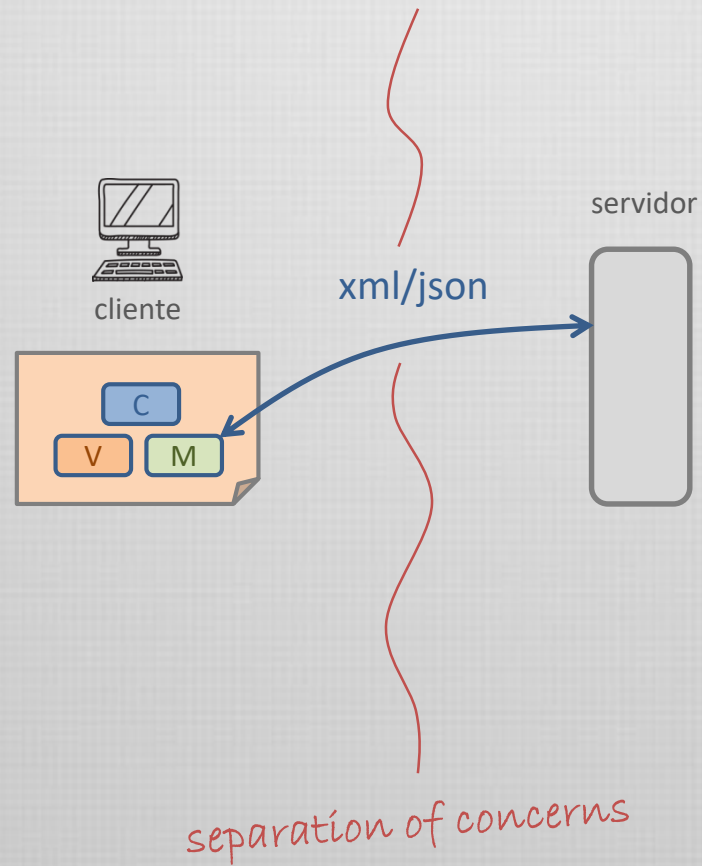


Ingeniería de Aplicaciones Web

Diego C. Martínez

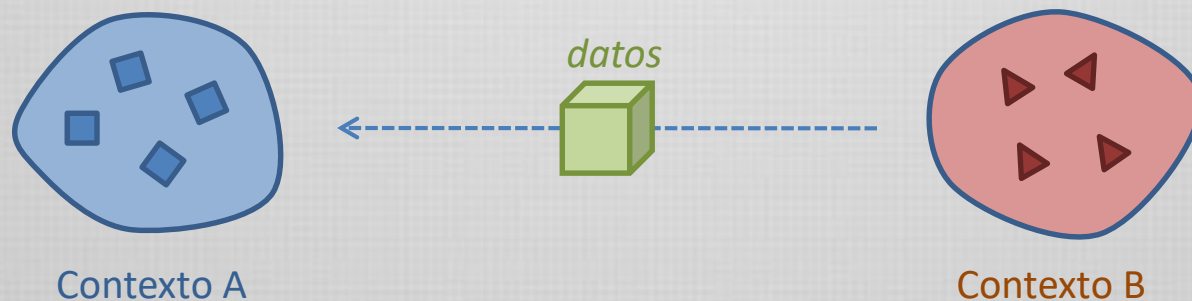
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Balance cliente - servidor



Interoperabilidad

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

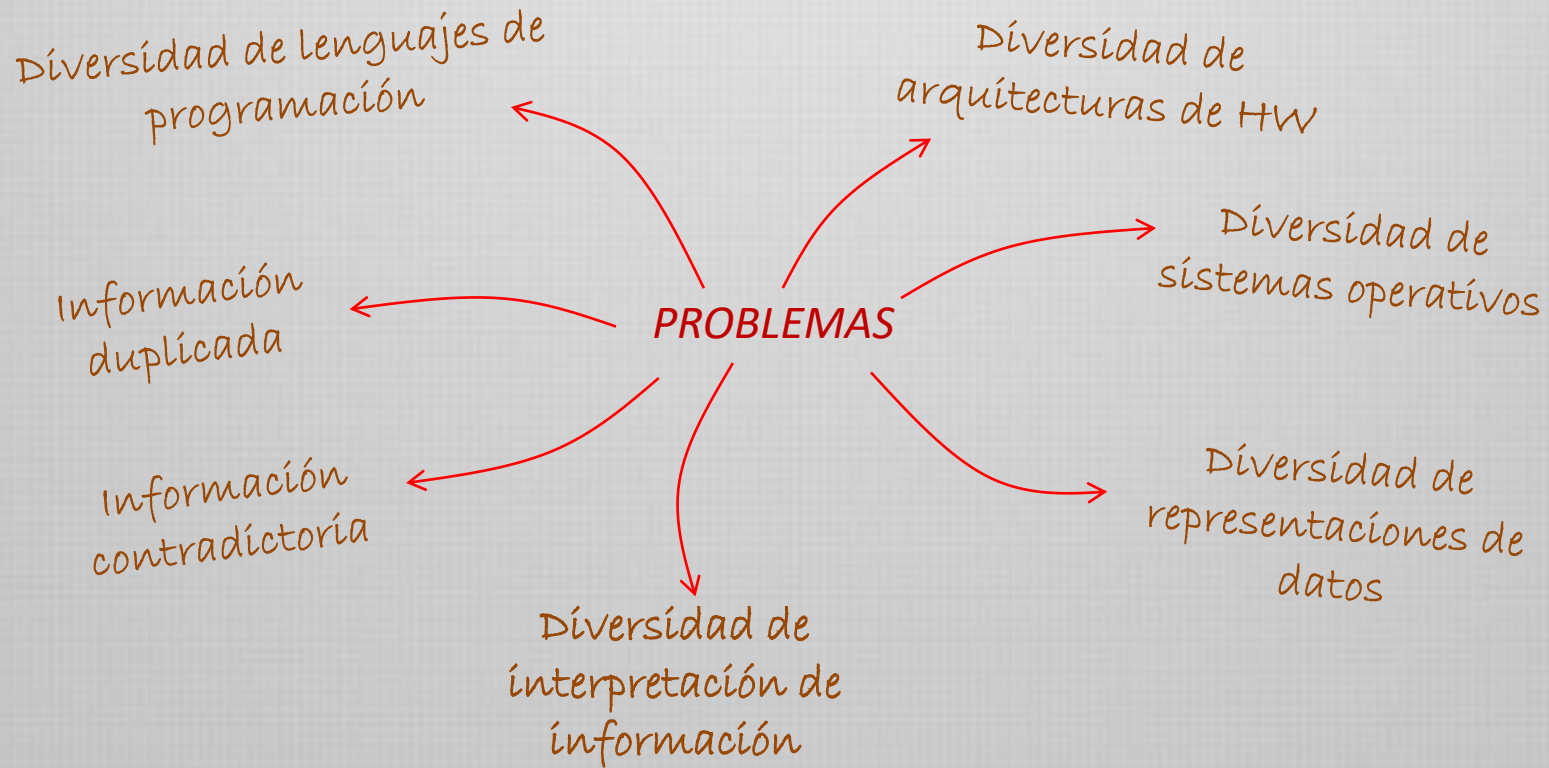


No es un concepto nuevo, ni es exclusivo de las tecnologías de información

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

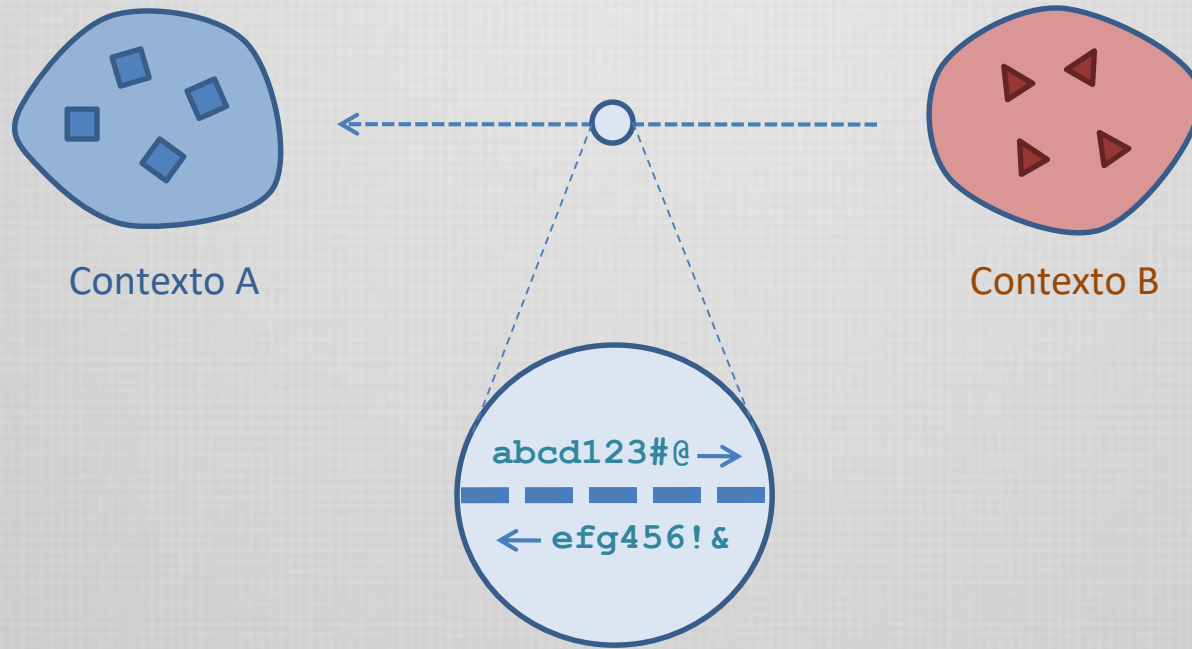
Interoperabilidad

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



“architectural mismatch”

Interoperabilidad

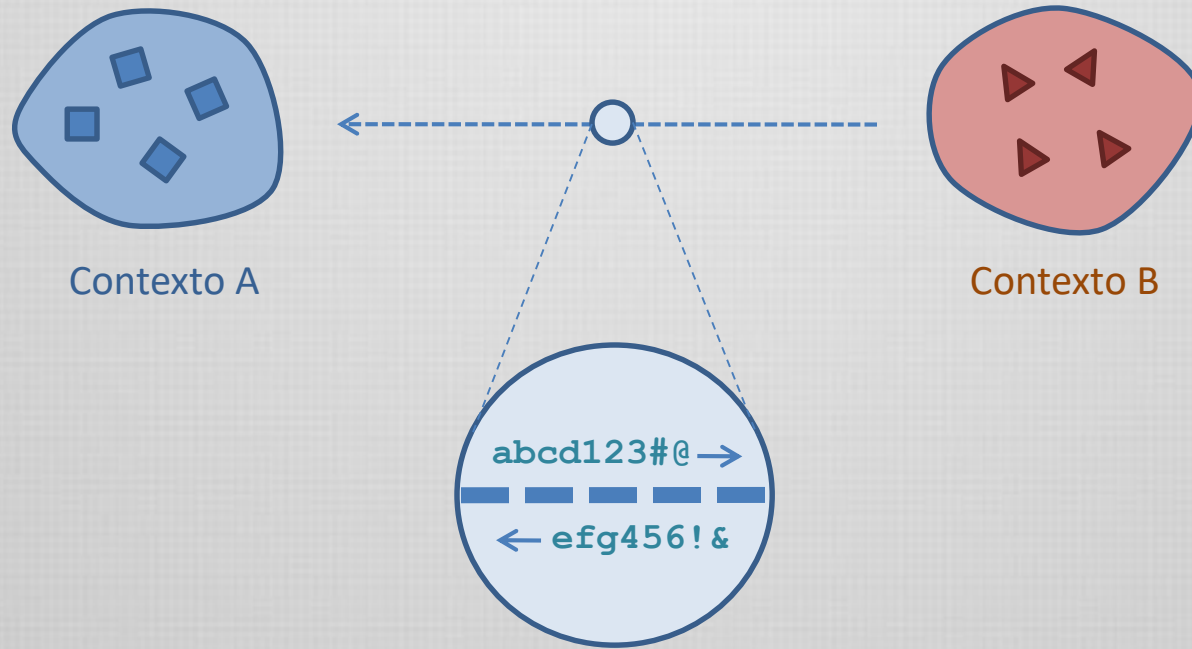


interoperabilidad sintáctica

“...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

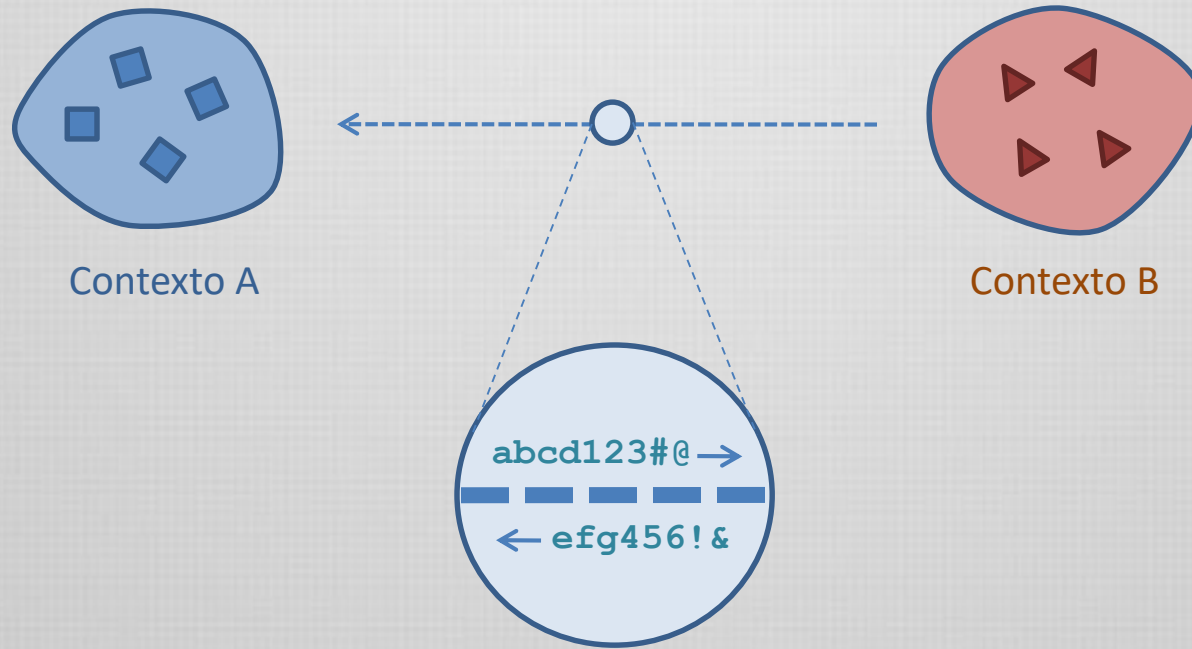


interoperabilidad sintáctica

“...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

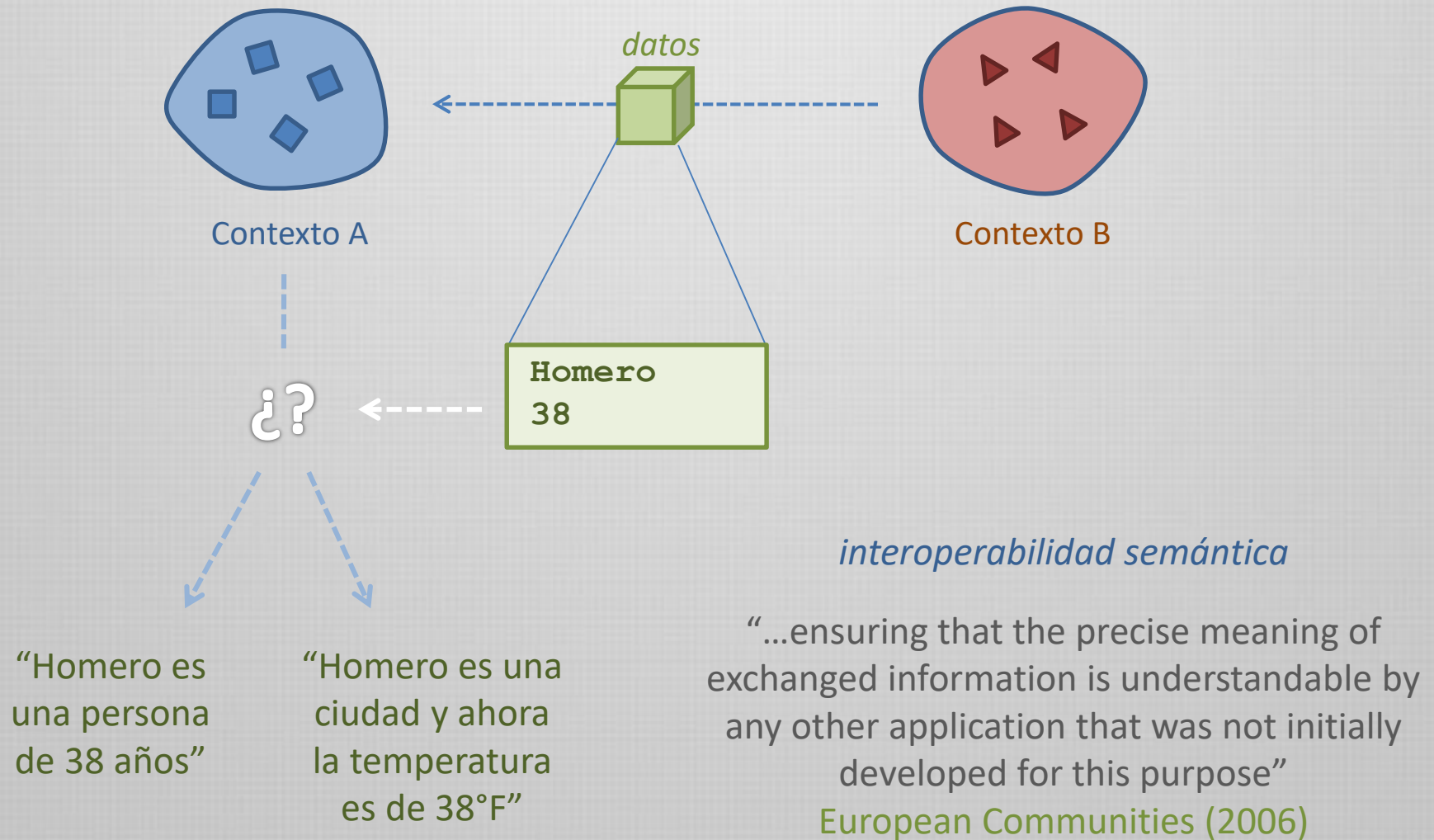


interoperabilidad sintáctica

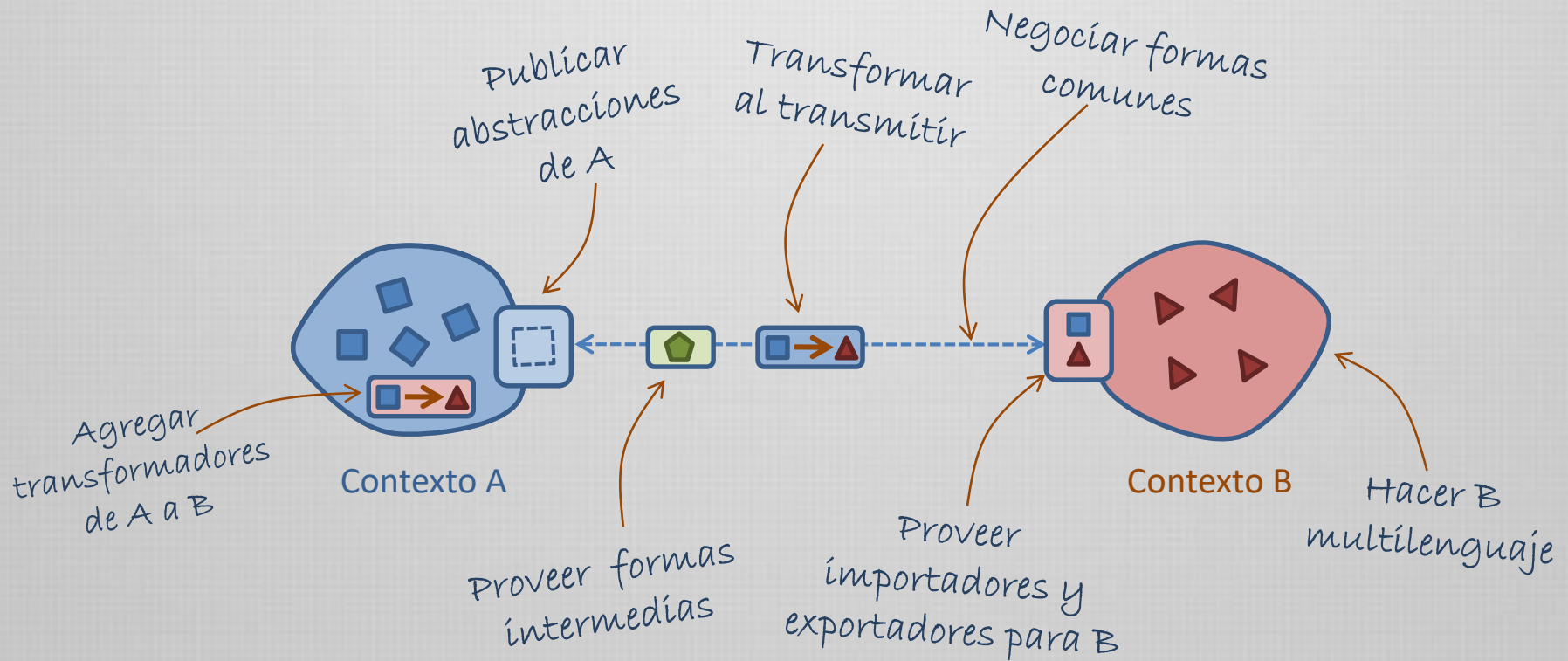
Desde la creación de [Internet](#), la interoperabilidad sintáctica no es una barrera para la operación

allí tenemos protocolos, estándares, seguridad, etc

Interoperabilidad

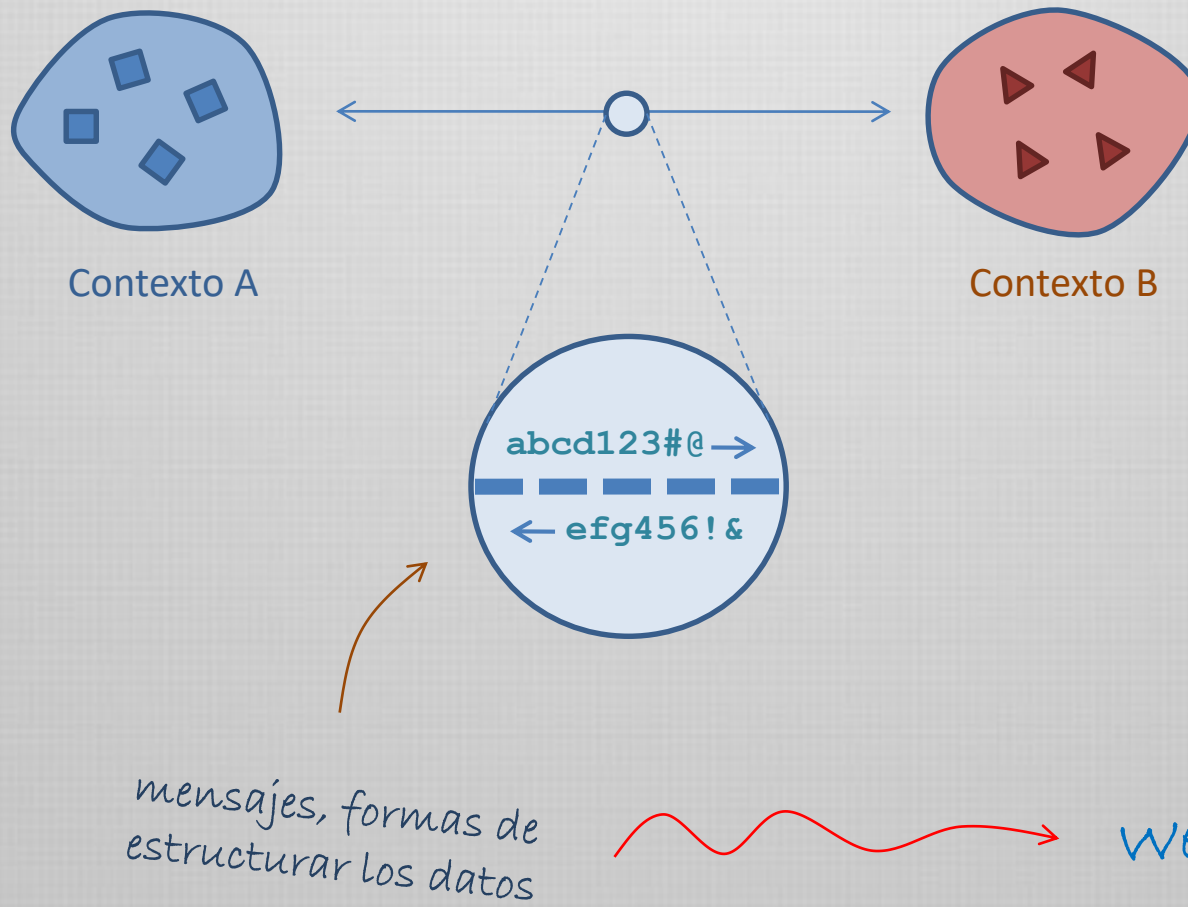


Architectural Mismatch – alternativas



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).

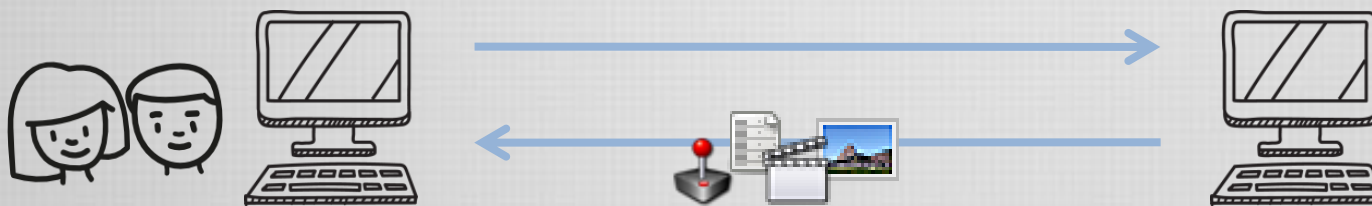
Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards

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.

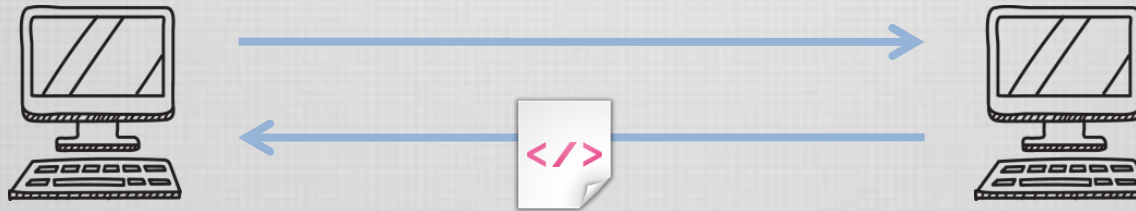
No tiene dependencias de ningún sistema operativo o lenguaje de programación.

La web para los humanos



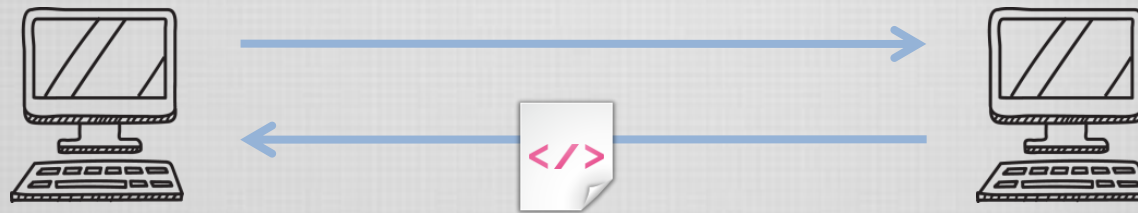
La web es en realidad una **red de servicios centrados en el humano**.
El consumidor final de los recursos de la web es el usuario humano.

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

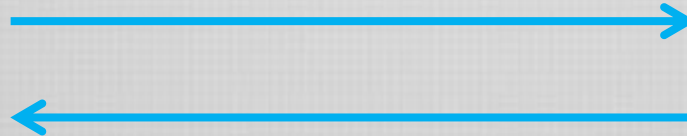
XPath

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>
```

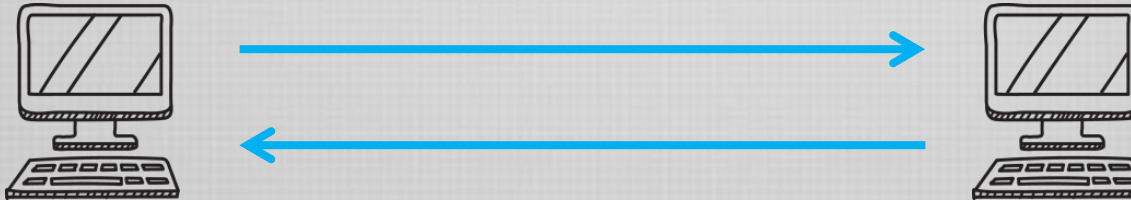


```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><double>18.24668429131</double></value>
    </param>
  </params>
</methodResponse>
```

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>
```

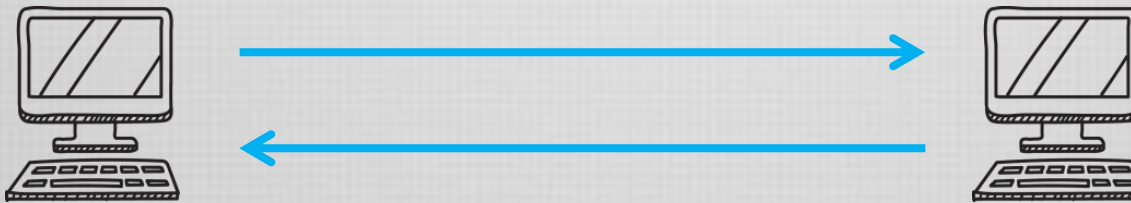


```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:getTempResponse xmlns:ns1="urn:xmethods-Temperature"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



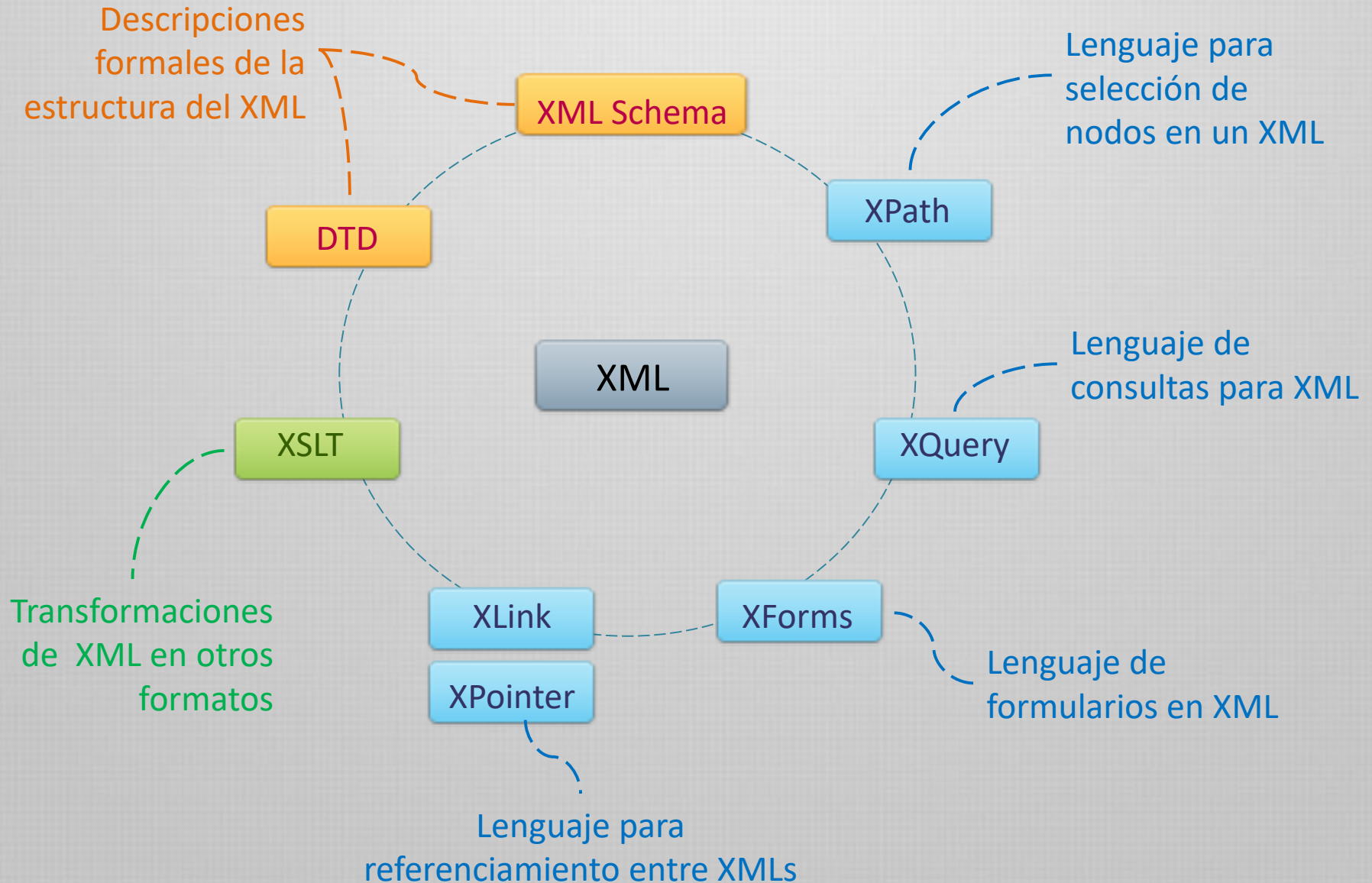
```
<?xml version='1.0' encoding='UTF-8'?>
<alumnos>
  <alumno>
    <nombre>Millhouse</nombre>
    <lu>12345</lu>
  </alumno>
  <alumno>
    <nombre>Bart</nombre>
    <lu>67890</lu>
  </alumno>
  <alumno>
    <nombre>Nelson</nombre>
    <lu>24680</lu>
  </alumno>
</alumnos>
```

Create
Retrieve
Update
Delete

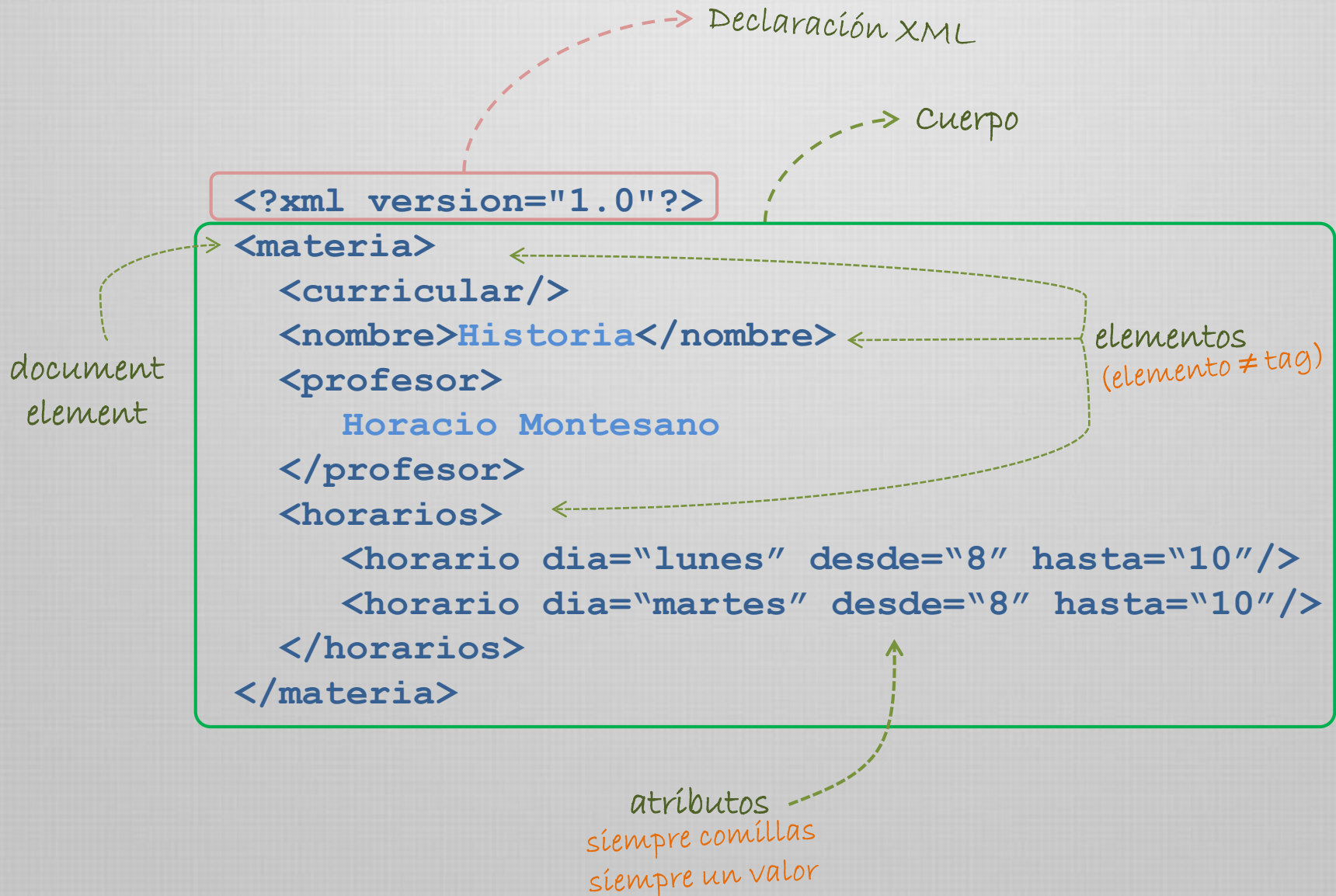
GET
POST
PUT
DELETE
HEAD

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" >
```

Puede definirse un espacio de nombres por defecto:

```
xmlns="http://otrohost.com/contactos"
```

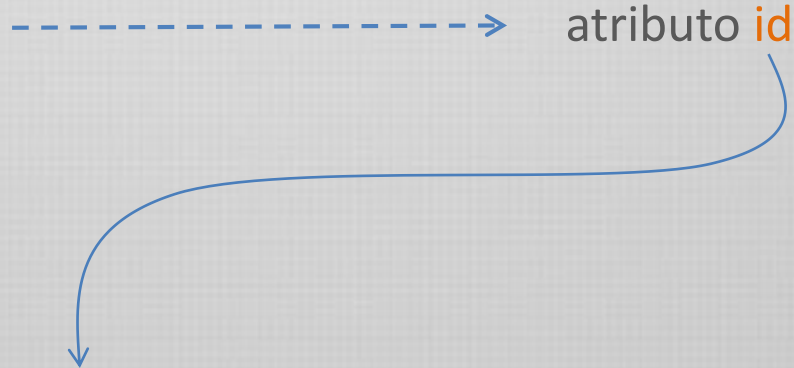
XML - namespaces

Espacios de nombres reservados:

- **xml** correspondiente a <http://www.w3.org/XML/1998/namespace>
- **xmlns** correspondiente a <http://www.w3.org/2000/xmlns>

Tampoco debe haber nombres de espacio que utilicen *xml* como prefijo.

Para identificar unívocamente
elementos individuales



```
<notebook id="HP23454" >
```

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

XML - validez

DTD – Document Type Definition

Descripción simple de qué elementos y atributos pueden existir en el XML

NO utiliza la misma sintaxis que XML.

Simple, pero con algunas limitaciones.
e.g, no es posible restringir valores a enteros.

XML Schema

Descripción de qué elementos y atributos pueden existir en el XML

XSD y XML se almacenan separados

Más poderoso que DTD, pero más complejo.
e.g, es posible definir tipos numéricos con subrangos, o el orden de los elementos dentro de un nodo.

Utiliza la misma sintaxis que XML

DTD- Document Type Definition

El DTD describe la estructura del documento.

Puede incluirse en el mismo documento o en un recurso externo.

internal subset

```
<!DOCTYPE nombre_nodo_raiz [  
    definiciones  
>
```

external subset

```
<!DOCTYPE nombre_nodo_raiz SYSTEM URI_DTD >
```

```
<!DOCTYPE nombre_nodo_raiz PUBLIC DTD_NAME URI_DTD >
```

DTD_NAME

"prefix//owner_DTD//descripcion_DTD//ISO 639_language_id"

prefix	{	ISO	Estándar ISO
		+	Estándar no-ISO aprobado
		-	Estándar no-ISO no aprobado

```
<!DOCTYPE document SYSTEM "autopartes.dtd">
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"  
    "http://www.w3.org/TR/REC-html40/loose.dtd">
```

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"/>
```


XSD – XML Schema Definitions

Se definen varios tipos de datos básicos para los elementos del XML:

```
xs:string  
xs:decimal  
xs:integer  
xs:boolean  
xs:date  
xs:time  
xs:anyURI
```

Es posible definir nuestros propios tipos de datos

Pueden ser tipos complejos o tipos simples con restricciones de valores

```
<xs:simpleType name="nombre_tipo">  
    <!-- definiciones -->  
</xs:simpleType>
```

```
<xs:complexType name="nombre_tipo">  
    <!-- definiciones -->  
</xs:complexType>
```

XSD – XML Schema Definitions

Para los tipos simples, pueden definirse restricciones, listas o uniones sobre *built-in types*.

xsd:restriction -----> *El nuevo tipo es una restricción de un tipo existente*

xsd:list -----> *El nuevo tipo es una lista de valores de otro tipo simple*

xsd:union -----> *El nuevo tipo es una unión de dos o más tipos simples*

Restricciones posibles:

xsd:enumeration	Conjunto de valores válidos.
xsd:fractionDigits	Máxima cantidad de dígitos decimales
xsd:length	Longitud (caracteres, bytes o items).
xsd:maxExclusive	Valor máximo (excluido).
xsd:maxInclusive	Valor máximo (incluido).
xsd:maxLength	Máxima longitud (caracteres, bytes o items)
xsd:minExclusive	Valor mínimo (excluido)
xsd:minInclusive	Valor mínimo (incluido).
xsd:minLength	Mínima longitud (caracteres, bytes o items).
xsd:pattern	Patrón basado en expresión regular.
xsd:totalDigits	Máxima cantidad de dígitos de un número.
xsd:whiteSpace	Reglas para el tratamiento de los whitespaces.

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>
```


XSD – XML Schema Definitions - tipos complejos

```
<xsd:complexType name="tipoSocio">
  <xsd:sequence>
    <xsd:element name="nombre" type="xsd:string"/>
    <xsd:element name="telefono" type="tns:Phone"/>
  </xsd:sequence>
  <xsd:attribute name="nsocio" type="xsd:integer"/>
</xsd:complexType>

<xsd:element name="socio" type="tipoSocio"/>
```

Ejemplo válido según la regla anterior:

```
<socio nsocio="123321">
  <nombre>Homero Simpson</nombre>
  <telefono>(123) 456-789</telefono>
</socio>
```

XSD – XML Schema Definitions - tipos complejos

También es posible definir extensiones entre tipos...

```
<xs:complexType name="DireccionARG">
  <xs:complexContent>
    <xs:extension base="Direccion">
      <xs:sequence>
        <xs:element name="provincia" type="xs:string"/>
        <xs:element name="codigopostal" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XPath

XPath es un lenguaje para referenciar partes de un documento XML.

Fuertemente relacionado con otros estándares como XSLT, XQuery y XPointer

<http://www.w3.org/TR/xpath/>

Utiliza una sintaxis simple que permite integrarlo en otras tecnologías

XPath modela un documento XML como un árbol de nodos.

nodos elemento

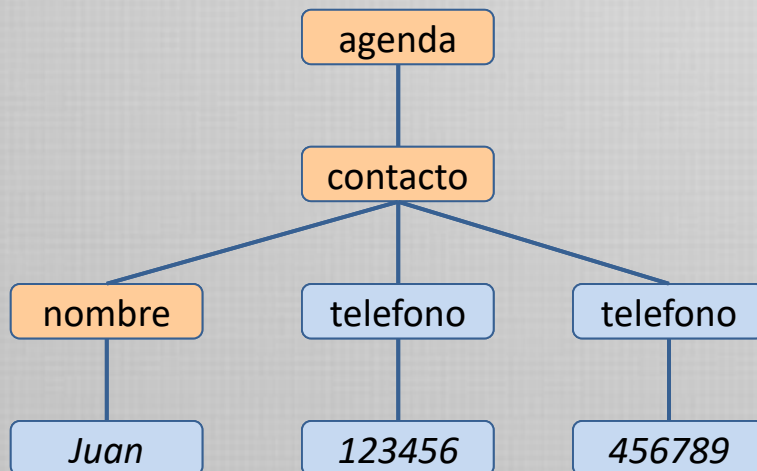
comentarios

nodos atributo

instrucciones de procesamiento

nodos texto

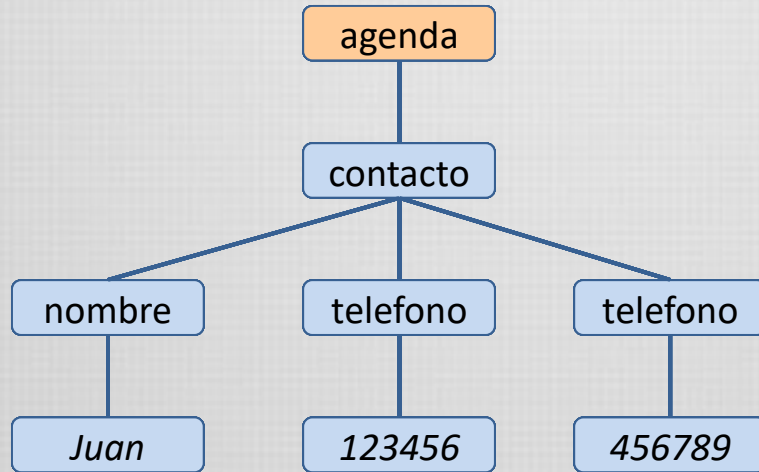
Relaciona strings (*paths*) con nodos del árbol del documento XML



agenda/contacto/nombre

Se pueden referenciar nodos individuales o grupos de nodos.

XPath



agenda/contacto/telefono

agenda/contacto/telefono[1]

agenda/contacto

/agenda

XPath- expresiones

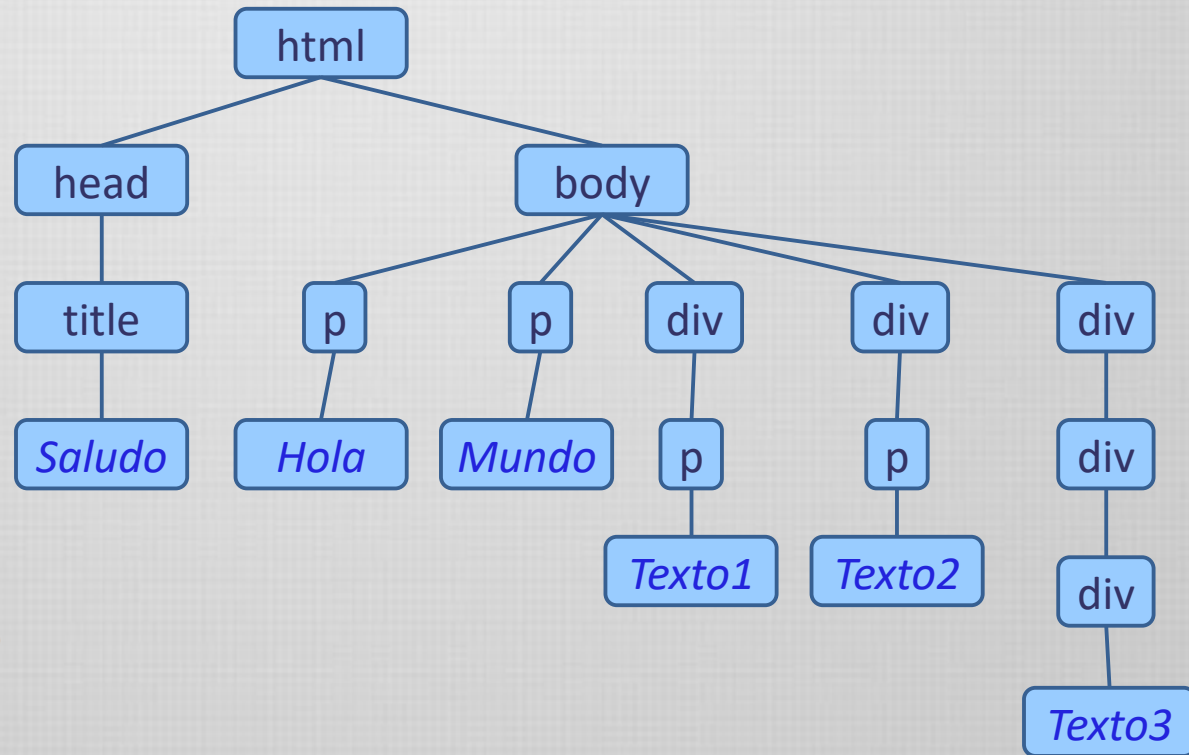
Una expresión en Xpath es evaluada a un objeto, que puede ser

- un conjunto de nodos
- un booleano
- un número
- un string

Expresión	Significado
/	<i>Referencia desde el nodo raíz</i>
/nodoX	<i>Encontrar el nodo raíz llamado <i>nodoX</i></i>
//elementoA	<i>Seleccionar los nodos desde el nodo actual, cualquiera sea el path</i>
.	<i>El nodo actual</i>
..	<i>El padre del nodo actual</i>
@atribN	<i>Selecciona el atributo <i>atribN</i> del nodo actual</i>
elemento[N]	<i>Selecciona el elemento N-ésimo</i>
elemento[@atribN]	<i>Selecciona los elementos que tengan el atributo <i>atribN</i></i>
elemento[@atribN='valorN']	<i>Selecciona los elementos cuyo valor de atributo <i>atribN</i> sea <i>valorN</i></i>
text()	<i>Selecciona el texto del elemento actual</i>

XPath- expresiones

```
<html>
<title>Saludo</title>
<body>
  <p>Hola</p>
  <p>Mundo</p>
  <div><p>Texto1</p></div>
  <div><p>Texto2</p></div>
  <div><div><div>
    Texto3
  </div></div></div>
</body>
</html>
```



html/head/title

html/head/title/text()

/html/body/p[2]

/html//p

/html//p[1]

//div

//div[1]

div/div

/div/div

//p/text()

//div/../p

XPath- expresiones

Otras expresiones interesantes:

```
/libro/articulo[5]/seccion[3]
```

```
libro/*/apellido
```

```
//div[@class='noticia']/p[@class='encabezado']/div
```

```
//div[last()]
```

```
/stock/producto[@tipo='oferta' and precio<50]
```

```
table[@border and @style]
```

```
//div[@*]
```

```
count(//div[@tipo='post'])
```

```
//table/@class
```

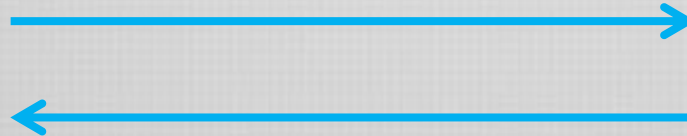
```
//graduado[@univ="UNS"]
```

XML-RPC

Web Service – XML-RPC

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

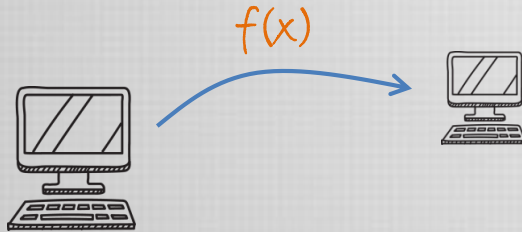
```
<?xml version="1.0"?>
<methodCall>
  <methodName>circleArea</methodName>
  <params>
    <param>
      <value><double>2.41</double></value>
    </param>
  </params>
</methodCall>
```



```
<?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.

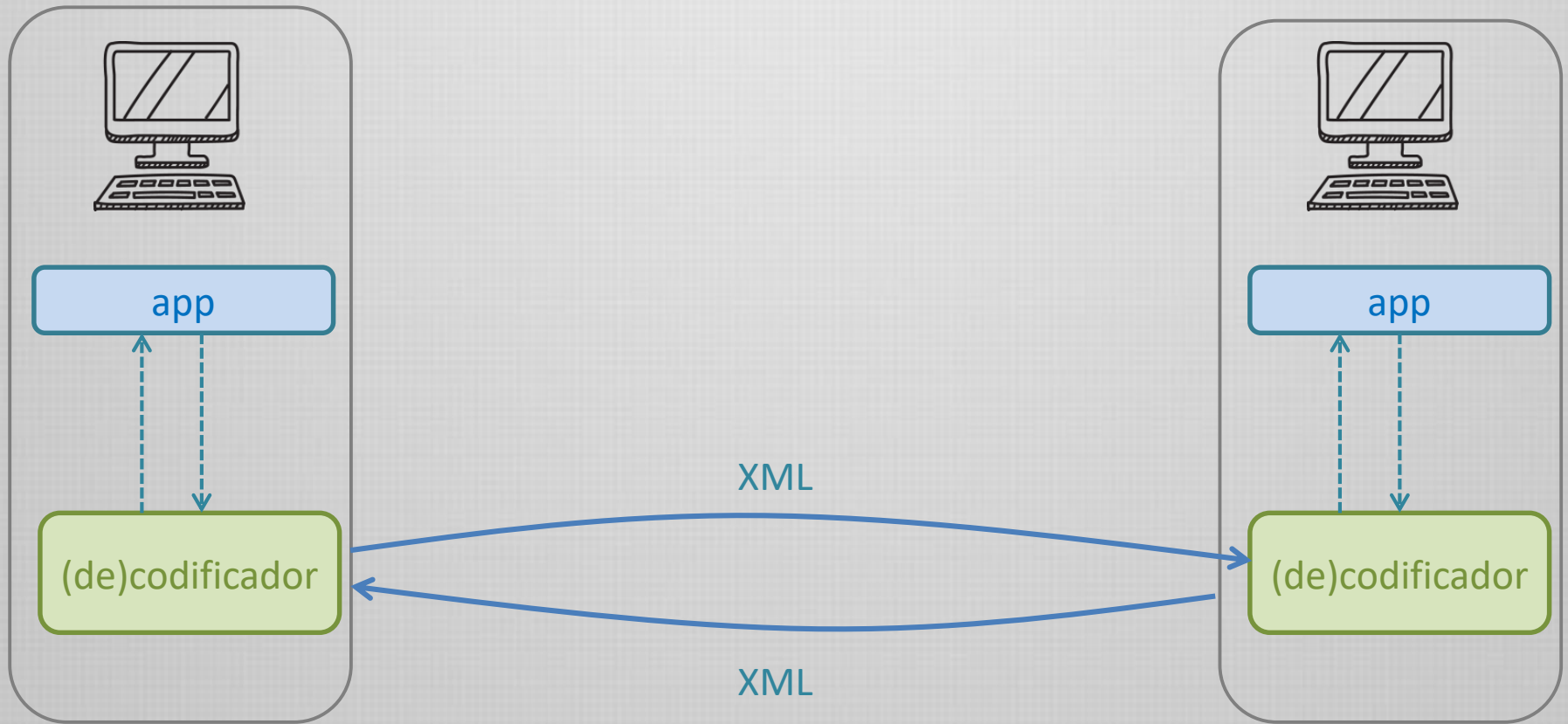
Para que la comunicación entre partes heteróneas pueda realizarse, la llamada y los datos deben formatearse apropiadamente (*marshalling*).

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"
);
```

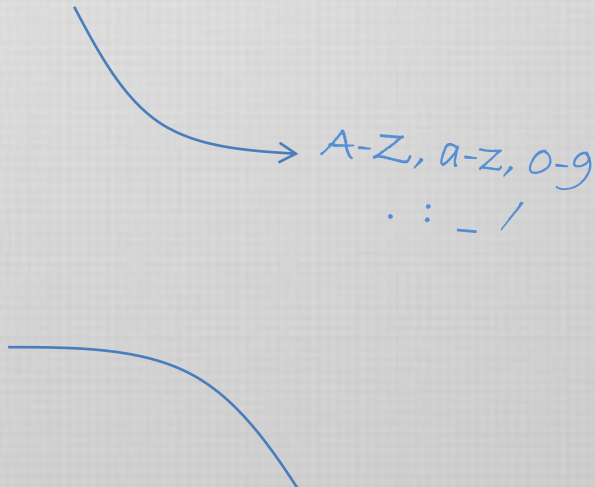
```
<value>
  <struct>
    <member>
      <name>nombre</name>
      <value>
        <string>Juan</string>
      </value>
    </member>
    <member>
      <name>deuda</name>
      <value>
        <double>23.5</double>
      </value>
    </member>
  </struct>
</value>
```

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>
```


Ejemplo implementación: XML-RPC en PHP

PHP ofrece librerías para utilizar XML-RPC con relativa facilidad.

Algunos métodos de la extensión *xmlrpc*:

`xmlrpc_encode($array)`

Codifica un arreglo de parámetros en el elemento *params*.

`xmlrpc_decode($xml)`

Decodifica un elemento params en datos de tipos nativos de PHP

`xmlrpc_encode_request($metodo, $array)`

Genera un request en XML. Se indica el método remoto y los parámetros

`xmlrpc_decode_request($xml, $metodo)`

Decodifica un mensaje de respuesta XML-RPC en datos de tipos nativos de PHP y devuelve en *\$metodo* el nombre del método invocado.

`xmlrpc_is_fault($datos)`

Devuelve verdadero si la respuesta decodificada es un error del servidor

Ejemplo implementación: XML-RPC en PHP

Algunos objetos de la extensión *PEAR*:

```
$val = new XML_RPC_Value(datos)
```

Crea un objeto que contiene valores XML_RPC

```
$val = new XML_RPC_Message($string, $array)
```

Crea un objeto mensaje. Recibe el nombre del método y los parámetros (un arreglo de objetos **XML_RPC_Value**)

```
$val = new XML_RPC_Client(path, server , port , proxy ...)
```

Crea un objeto cliente de un servicio XML-RPC. Es configurado con el servidor remoto y se encarga de toda la comunicación :)

Posee un método send para enviar un **XML_RPC_Message**

```
XML_RPC_Response
```

Los objetos que devuelve la operación send de **XML_RPC_Client** son de tipo **XML_RPC_Response**.

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)

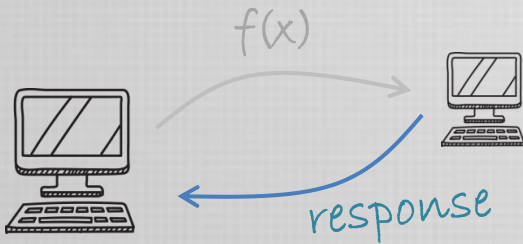
Una **notificación** es un objeto request sin "id"
El servidor no debe responder a las notificaciones.

```
{  
  "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  
}
```

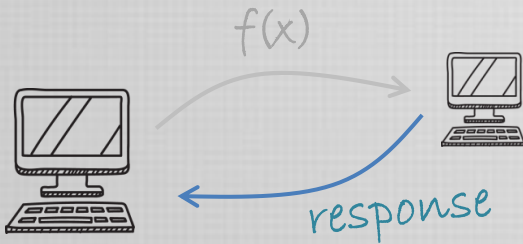


```
{  
  "jsonrpc": "2.0",  
  "result": 65,  
  "id": 65  
}
```

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"
  }
]
```

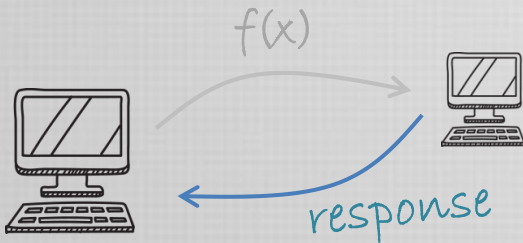


```
[
  { "jsonrpc": "2.0",
    "result": 7,
    "id": "1"
  },
  { "jsonrpc": "2.0",
    "result": 6,
    "id": "2"
  },
  { "jsonrpc": "2.0",
    "error": {
      "code": -32600,
      "message": "Invalid Request"
    },
    "id": null
  },
  { "jsonrpc": "2.0",
    "result": 2,
    "id": "5"
  }
]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

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>