Tema 4 – Formatos de serialización de datos (I).

Daniel Sánchez Martínez <danielsm@um.es>

Contenidos

- XML
  - **Origen**
  - Sintaxis
  - DTDs
  - XML Schema
  - Procesamiento

XML: eXtended Markup Language

- Los documentos electrónicos tienen 3 partes:
  - Contenido: los datos o palabras en sí mismos
  - Estructura: la organización del contenido
  - Presentación: el aspecto con el que se presenta el contenido al lector
- Las marcas son *secuencias de símbolos que, insertados en el contenido de un documento, sirven para indicar su presentación o su estructura.*

Con XML: las marcas indican la estructura

<Factura>
<De>Pepe Martinez**</De>**
<A>Juan Fernandez**</A>**
<Fecha año=2000 mes=12 dia =1**/>**
<Cantidad moneda=Pts>1000**</Cantidad>**
<Iva>17**</Iva>**
<Total>1170**</Total>**
</Factura>

<Factura> : comienza la información de factura
<De> ... **</De>** : Quién creó la factura

Origen XML

- Hacia finales de los sesenta, un grupo de investigadores comenzó a interesarse por darle otros usos a los documentos electrónicos. En particular, IBM pidió a **Charles Goldfarb** que construyese un sistema para almacenar, buscar, gestionar y publicar documentos legales.

- El resultado de su trabajo fue el lenguaje **SGML** (Standard Generalized Markup Language), actualmente el estándar de la ISO 8879. **XML** es un subconjunto de SGML pensado para ser llevado al web.

Origen XML

- Goldfarb y otros investigadores de IBM reconocieron tres hechos importantes:
  - Para intercambiar información, los programas tienen que soportar un **lenguaje común**. Tiene sentido que ese lenguaje común sea algún tipo de lenguaje de marcas.
  - **La estructura de un documento se puede ver como una jerarquía de elementos**. Por ejemplo, una carta puede tener, en un primer nivel, un elemento "encabezado" y otro "cuerpo". El encabezado puede contener, a su vez, elementos "destinatario", "asunto", etc. Las marcas que delimitan estos elementos se denominan marcas estructurales generalizadas (*structural generalized markup*).
  - Los documentos tienen que seguir algún tipo de **reglas**, es decir, el lenguaje de marcas debería ser especificado de algún modo formal que permitiese garantizar que el documento cumple cierta estructura.

Origen XML

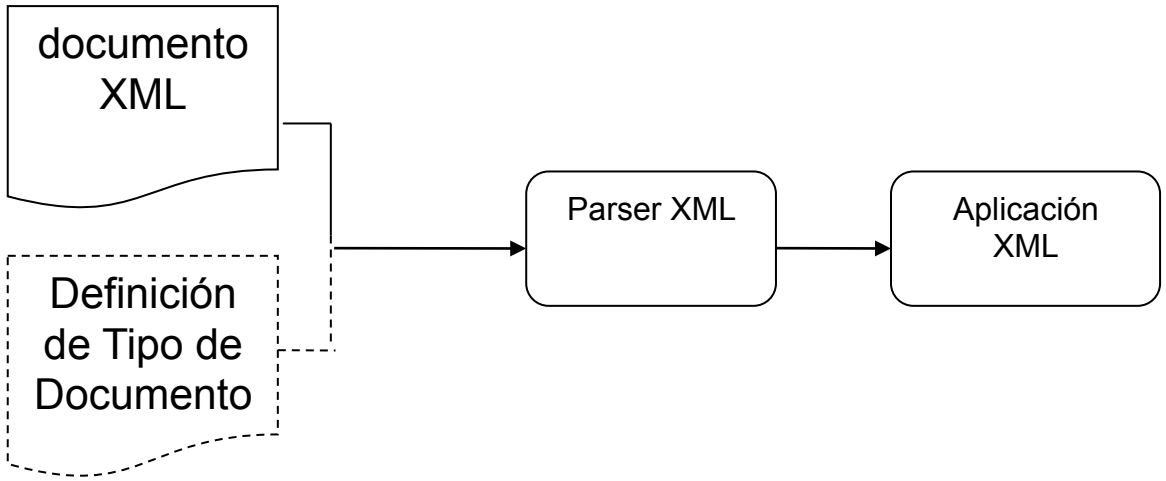
- Una vez que se tiene definido un tipo de documento, es necesario definir cómo se va a imprimir o a visualizar en una pantalla. Esta descripción se recoge en una ***hoja de estilo***.
- La idea central de la solución propuesta por Goldfarb es que mantener separados los tres aspectos de un documento, contenido, estructura, y presentación, aporta grandes ventajas:
  - Permite al navegador hacer el trabajo de presentación de los datos en la pantalla
  - Facilita al navegador la manipulación de los datos, y la realización de cálculos a partir de ellos
  - Se pueden realizar búsquedas "inteligentes" de información.
  - Fomenta la creación de vocabularios (marcas) estándar en diferentes dominios, como los bancos, las telecomunicaciones, el transporte, etc.

Contenidos

- XML
  - Origen
  - **Sintaxis**
  - DTDs
  - XML Schema
  - Procesamiento

XML básico

- La siguiente figura muestra los elementos básicos de un sistema basado en XML:



Detalles sintácticos

- Los documentos XML están compuestos por caracteres del conjunto *Unicode*. Cualquier secuencia de caracteres se denomina *string*. La sintaxis de XML describe la forma de combinar *strings* para crear documentos XML bien formados.
  - XML distingue mayúsculas y minúsculas, y por tanto no es lo mismo "ELEMENT" que "element".
  - Las marcas se parecen a las de HTML. Empiezan por '<' o por '&'.
  - Los blancos son el espacio (ASCII 32), el tabulador (ASCII 9), el retorno de carro (ASCII 13) y el carácter de línea nueva (ASCII 10).
  - Los literales aparecen rodeados de comillas simples o dobles.
  - Los nombres que se pueden utilizar para designar elementos XML deben empezar por letra, '.' o '_', e ir seguidos de letras, dígitos, '-', '_', '.' y ':'.

Prólogo de un documento XML

- Los documentos XML están divididos en dos partes principales: un prólogo y una instancia de documento.
- El prólogo está compuesto de una declaración XML y una declaración de tipo de documento, y ambas son opcionales. Este es un ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE DOCBOOK SYSTEM "http://www.um.es/docbook">
```

- Este prólogo dice que el documento se ajusta a XML versión 1.0, y es una instancia de un determinado tipo de documento, DOCBOOK.

Contenido de un documento XML

- El contenido real del documento XML se encuentra en la instancia del documento. Este es un ejemplo de documento XML:

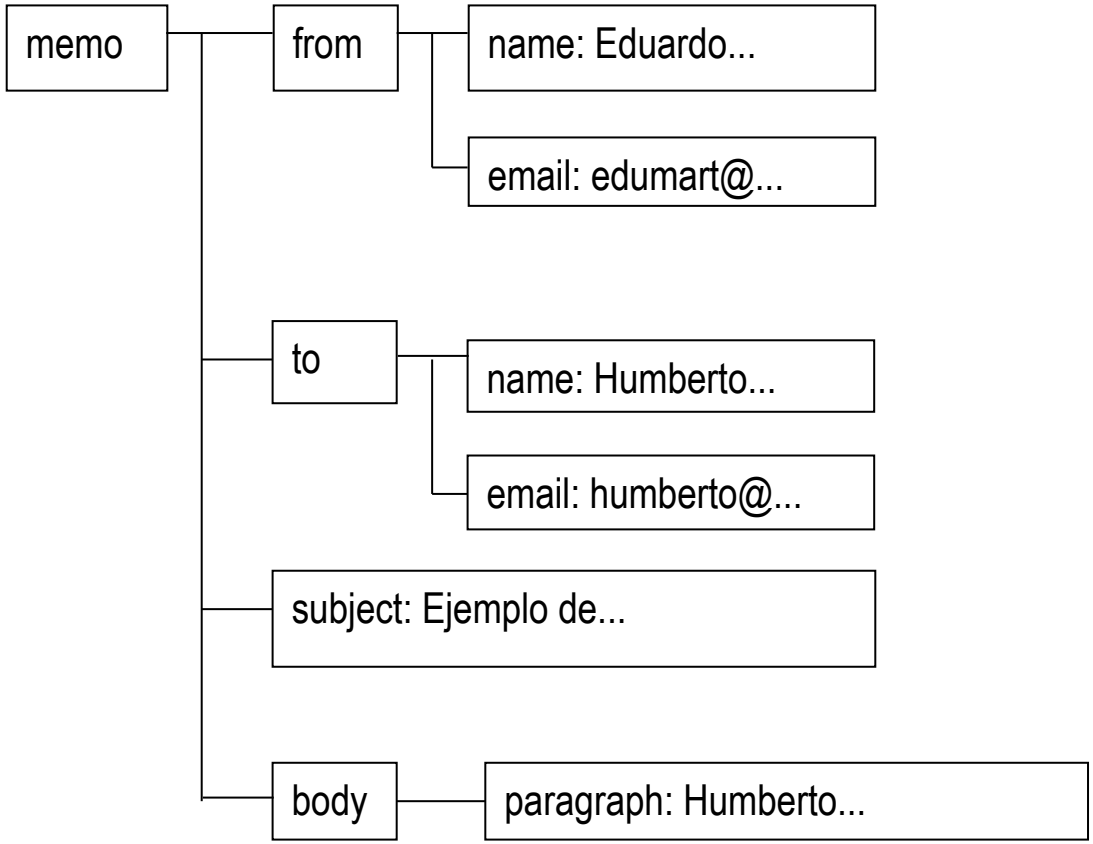
```
<?xml version="1.0"?>
<!DOCTYPE memo SYSTEM "memo.dtd">
<memo>
<from><name>Eduardo Martinez</name><email>edumart@um.es</email>
</from>
<to><name>Humberto Martinez</name><email>humberto@um.es</email>
</to>
<subject>Ejemplo de Memo</subject>
<body>
<paragraph>Humberto, te aseguro que <emphasis>no</emphasis> quería usar el ejemplo
    típico de Memo, pero no se me ocurría otra cosa.
</paragraph>
</body>
</memo>
```

prólogo

instancia de documento

Árbol sintáctico

- Parser XML: genera el árbol sintáctico



Elementos y atributos XML (1)

- Los elementos XML se dividen en dos categorías: los que tienen contenido y los que son vacíos. Ejemplo de elemento con contenido:

```
<titulo>Este es el título</titulo>
```

- Los elementos XML sin contenido, o elementos vacíos, tienen este aspecto:

```
<Fecha año="1999" mes="2" dia="1" />
```

- Los atributos son una forma de adjuntar características o propiedades a los elementos de un documento. Los atributos tienen nombre y valor.

Elementos y atributos XML (2)

- A veces, un elemento con subelementos puede ser modelado de forma equivalente con un elemento vacío con atributos. A continuación se dan tres formas de modelar un elemento persona de un mensaje de correo:

```
<FROM>
  <NAME>Eduardo Martinez</NAME>
  <EMAIL>edumart@um.es</EMAIL>
</FROM>

<FROM NAME="Eduardo Martinez" EMAIL="edumart@um.es" />

<FROM EMAIL="edumart@um.es">
Eduardo Martinez
</FROM>
```

Espacios de nombres (1)

- Los vocabularios de marcas se deben **reutilizar**
- Un mismo documento puede usar varios vocabularios de marcas
  - Posibilidad de **colisión entre nombres** de elementos y atributos
  - Solución: **nombres universales**
- **Definición**: un **espacio de nombres XML** es una colección de nombres, identificada con una referencia URI, que se emplean en documentos XML como nombres de elementos y nombres de atributos.
- **Definición**: dos referencias URI que identifican espacios de nombres se consideran idénticas si son iguales carácter a carácter.
- Los nombres de elementos y atributos aparecen **cualificados:**

prefijo:parte_local

- El prefijo corresponde a una referencia URI, que identifica el espacio de nombres

Espacios de nombres (2)

- El espacio de nombres se declara con un atributo especial de un elemento XML:
  - El atributo de la forma '***xmlns:NCName=URI',*** declara que el elemento en el que aparece pertenece al espacio de nombres URI, y se usa el prefijo *NCName* para cualificar los nombres de elementos y atributos
  - Un elemento puede tener varias declaraciones de espacios de nombres.
  - Ejemplo: espacio de nombres http://ecommerce.org/schema con prefijo *edi*

```
<edi:x xmlns:edi='http://ecommerce.org/schema'>
<!-- El prefijo edi se liga a la URI en el elemento x
  y su contenido -->
  <edi:y>...</edi:y>
  <z edi:att="XX">...</z>
</edi:x>
```

Espacios de nombres (3)

- **Espacio de nombres por defecto**: espacio de nombres declarado sin prefijo. Por defecto se considera que cualquier nombre sin prefijo pertenece a dicho espacio de nombres. Ejemplo:

```
<?xml version="1.0"?>
<book xmlns="http://www.um.es/books"
      xmlns:isbn="http://www.isbn.org/isbn">
  <title>Manual de XML</title>
  <isbn:number>1234566789</isbn:number>
  <notes>
      <p xmlns='http://www.w3c.org/HTML'>
            Este es un <i>gran</i> libro
      </p>
  </notes>
</book>
```

Contenidos

- XML
  - Origen
  - Sintaxis
  - **DTDs**
  - XML Schema
  - Procesamiento

Document Type Definition (DTD)

- **Documento bien formado**: documento que cumple las reglas sintácticas de XML.
- Es interesante definir los tipos de elementos permitidos, atributos y entidades, y puede expresar restricciones sobre sus combinaciones válidas: se consigue con DTDs
- **Documento válido**: documento que declara en su prólogo cierto DTD, y que efectivamente lo cumple.
- **Documento no válido**: documento que declara en su prólogo cierto DTD, y que no lo cumple.
- Un documento no válido puede ser bien formado
- Un documento bien formado puede no ser ni válido ni no válido

DTD interno

```
<?xml version="1.0"?>
<!DOCTYPE label[
     <!ELEMENT label (name, street, city, state, country, code)>
     <!ELEMENT name (#PCDATA)>
     <!ELEMENT street (#PCDATA)>
     <!ELEMENT city (#PCDATA)>
     <!ELEMENT state (#PCDATA)>
     <!ELEMENT country (#PCDATA)>
     <!ELEMENT code (#PCDATA)>
]>
<label>
<name>Rock N. Robyn</name>
<street>Jay Bird Street</street>
<city>Baltimore</city>
<state>MD</state>
<country>USA</country>
<code>43214</code>
</label>
```

DTD externo

- Si se indica "SYSTEM", a continuación aparece una URL que localiza el fichero que contiene el DTD.

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

<!DOCTYPE LABEL SYSTEM "http://www.sgmlsource.com/
  dtds/label.dtd">

<LABEL>

...

</LABEL>
```

- Si se emplea "PUBLIC", se hace referencia al DTD externo mediante un identificador público único:

```
<!DOCTYPE MEMO PUBLIC "-//SGMLSOURCE//DTD MEMO//EN"
   "http://www.sgmlsource.com/dtds/memo.dtd">

<MEMO>...</MEMO>
```

DTD - Definición de elementos XML

- Los elementos son la base del lenguaje XML. Cada elemento de un documento XML válido debe conformar con un tipo de elemento declarado en el DTD.
- Las declaraciones de tipos de elementos deben comenzar con el string "<!ELEMENT", seguido del nombre (o *identificador genérico*) del tipo de elemento que se está declarando. Finalmente, debe aparecer una *especificación de contenido*.

```
<!ELEMENT memo (to, from, body)>
```

DTD - Contenido de elementos (1)

- **EMPTY:** indica que un elemento no puede tener contenido, como la etiqueta IMG de HTML:

```
<!ELEMENT elemento-vacio EMPTY>
```

- **ANY:** el elemento puede contener cualquier tipo de sub-elemento o datos (string de caracteres). Un elemento con la especificación de contenido ANY es completamente no estructurado.

```
<!ELEMENT cualquier-cosa ANY>
```

DTD - Contenido de elementos (2)

- #PCDATA se refiere a cualquier secuencia de caracteres que no contiene elementos.
- **Contenido mixto:** si deseamos que el contenido de un elemento esté formado por cierta combinación estructurada de datos y sub-elementos, es necesario emplear expresiones regulares (*, +, ?) para describir la sintaxis de dicho contenido:

```
<!ELEMENT parrafo (#PCDATA|enfatizado)*>
<!ELEMENT resumen (#PCDATA|enfatizado|cita)*>
```

DTD - Expresiones regulares

- Permiten definir una gramática de forma recursiva
- Las expresiones regulares más sencillas son:
  - (#PCDATA) : caracteres
  - (*elemento*): el contenido es una y sólo una aparición de *elemento*
- Expresiones regulares que indican cardinalidad:
  - (*ER*?) : contenido opcional, es decir, vacío o cualquier contenido descrito con la expresión regular *ER*.
  - (*ER**): contenido con cardinalidad máxima no definida y mínima posiblemente 0
  - (*ER*+): contenido con cardinalidad máxima no definida y mínima 1
- Expresión regular que indica una opción:
  - (*ER1* | *ER2*): contenido que puede ser cualquiera de los descritos por *ER1* y los descritos por *ER2*
- Expresión regular que indica secuencia:
  - (*ER1*, *ER2*, *ER3*, ...): contenido formado por la secuencia del contenido descrito por *ER1*, seguido por el descrito por *ER2*, ...

DTD - Atributos (1)

- Otra forma de añadir información a un elemento es definiendo atributos. La principal diferencia entre los atributos y el contenido de un elemento no vacío es que el valor de los primeros no puede contener sub-elementos.
- Los atributos se declaran para tipos de elementos específicos:

```
<!ELEMENT PERSONA (#PCDATA)>
<!ATTLIST PERSONA EMAIL CDATA #REQUIRED>
```

- Las declaraciones de atributos empiezan con el string "<! ATTLIST". Inmediatamente después de un espacio en blanco viene el identificador del elemento. Después se indica el nombre del atributo, su tipo y una indicación relativa a su valor por defecto.

DTD - Atributos (2)

- Los atributos pueden tener valores por defecto:

```
<!ATTLIST CAMISA TAMAÑO (PEQUEÑO|MEDIO|GRANDE) MEDIO>
```

- Los atributos pueden declararse de forma que sus valores cumplan ciertas restricciones de tipo léxico o semántico.
  - **CDATA**: significa "character data". El contenido es cualquier secuencia de caracteres que no contenga otros elementos.

```
<!ATTLIST PERSONA EMAIL CDATA #REQUIRED>
```

  - **Enumerados**: se emplea cuando un atributo debe tomar un valor dentro de un conjunto discreto:

```
<!ATTLIST PERSONA ELECCION (OPCION1|OPCION2|OPCION3)
#REQUIRED>
```

DTD - Atributos (3)

- **ID e IDREF**: permite crear referencias cruzadas entre elementos.

```
<!ELEMENT SECTION (TITULO, PARRAFO*)>
<!ATTLIST SECTION IDENTIFICADOR ID #REQUIRED>
<!ELEMENT REFERENCIA EMPTY>
<!ATTLIST REFERENCIA TARGET IDREF #IMPLIED>
...
<SECTION IDENTIFICADOR="Capitulo 1">
...
</SECTION>
...
Vea <REFERENCIA TARGET="Capitulo 1"/> para volver a
  leer sobre
...
```

DTD - Atributos (4)

- **ENTITY**: tipo de los atributos cuyo valor es el nombre de una entidad.

```
<!ATTLIST REFERENCIA-LIBRO TARGET ENTITY
  #REQUIRED>

...

<!ENTITY otro-libro SYSTEM "http://
  www.amazon.com/....">

...

<REFERENCIA-LIBRO TARGET="otro-libro">

...
```

DTD - Ejemplo completo (1)

- Listin.dtd

```
<!ELEMENT listin (persona)+>
<!ELEMENT persona (nombre, email*, relacion?)>
<!ATTLIST persona id ID #REQUIRED>
<!ATTLIST persona sexo (hombre | mujer) #IMPLIED>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT relacion EMPTY>
<!ATTLIST relacion amigo-de IDREFS #IMPLIED
    enemigo-de IDREFS #IMPLIED>
```

DTD - Ejemplo completo (2)

- Listin.xml

```
<?xml version="1.0"?>
<!DOCTYPE listin SYSTEM "listin.dtd">
<listin>

<persona sexo="hombre" id="ricky">
<nombre>Ricky Martin</nombre>
<email>ricky@puerto-rico.com</email>
<relacion amigo-de="leatitia">
</persona>

<persona sexo="mujer" id="leatitia">
<nombre>Leatitia Casta</nombre>
<email>castal@micasa.com</email>
</persona>

</listin>
```

Contenidos

- XML
  - Origen
  - Sintaxis
  - DTDs
  - **XML Schema**
  - Procesamiento

XML Schema (XSD)

- Alternativa a DTDs.
- Ventajas
  - XML Schema se basa en XML, lo cual permite validar los documentos.
  - Soporta una serie de tipos de datos (int, float, boolean, date, etc.).
  - Vocabulario abierto: permite definir nuevos tipos de datos y establecer nuevas relaciones de herencia.
  - Soporta espacios de nombres → asociar nodos individuales con las declaraciones de tipos de esquema.
  - Soporta grupos de atributos → combinación de atributos.

De DTD a XML Schema

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>To write Tove!</body>
</note>
```

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mysite.com"
xmlns="http://www.mysite.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

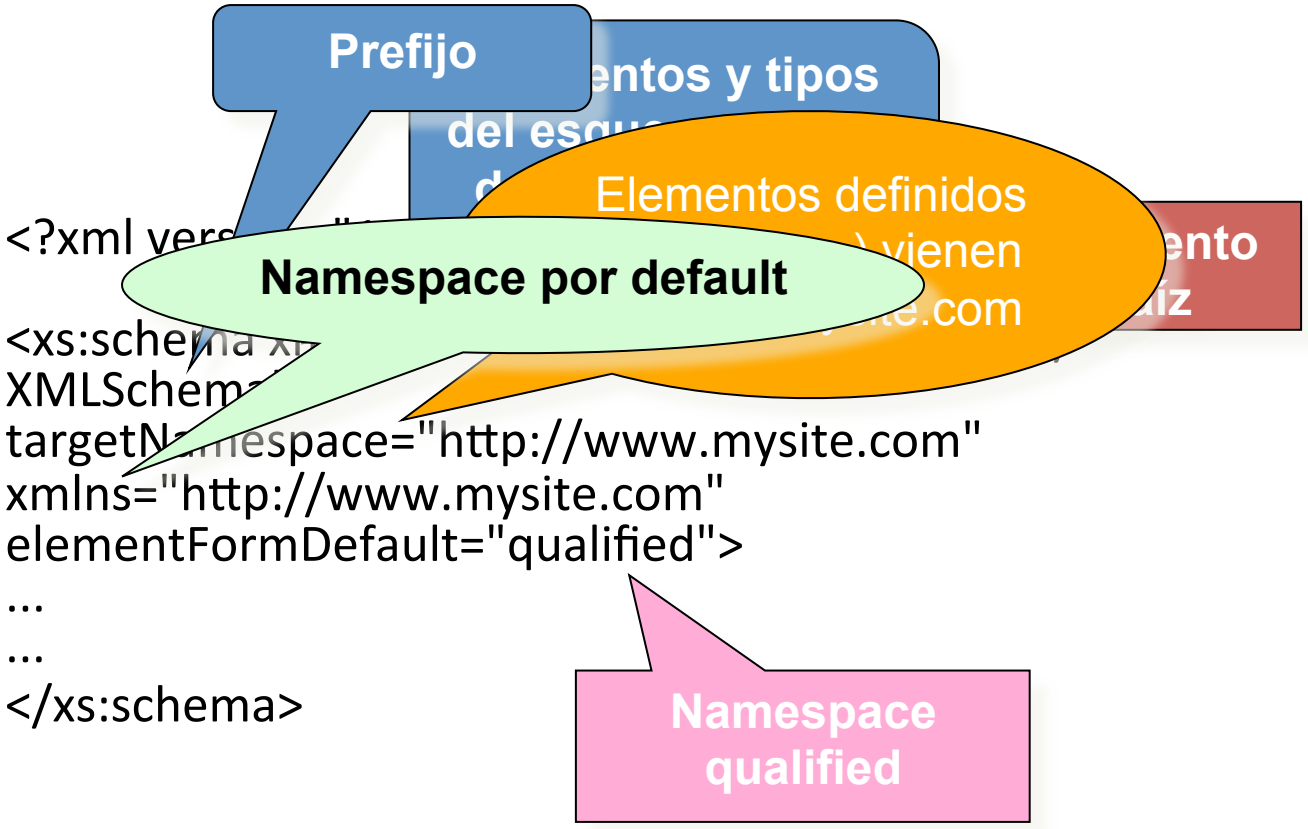
De DTD a XML Schema

- Referencia al archivo de definición:

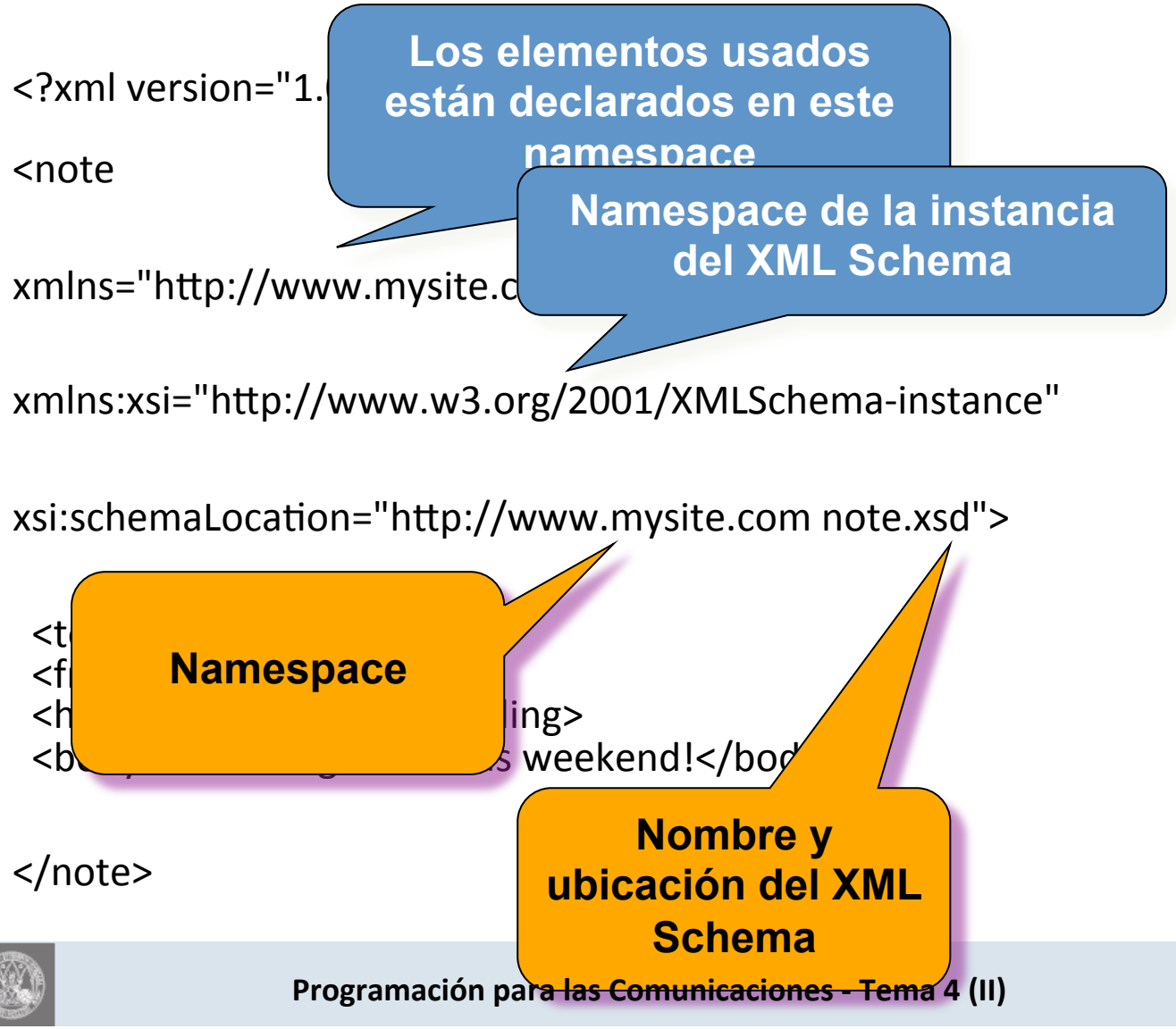
```
<?xml version="1.0"?>

<!DOCTYPE note SYSTEM
"http://www.mysite.com/dtd/note.dtd">

<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

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

<note
xmlns="http://www.mysite.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.mysite.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XSD - Declaraciones



XSD - Referencias



XSD - Elementos simples

- No puede contener otros elementos o atributos
- Puede contener únicamente "texto"
  - Tipos incluidos en la definición XML Schema (boolean, string, date, etc.), o
  - Un tipo personalizado que el usuario puede definir

XSD - Elementos simples

<xs:element name="xxx" type="yyy"/>

- Tipos más comunes:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

XSD - Elementos simples

```
<lastname>Aguilar</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Definición

XSD - Sintaxis de los Atributos

<xs:attribute name="xxx" type="yyy"/>

- Tipos más comunes:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

XSD - Elementos simples

Uso:

<lastname lang="EN">Smith</lastname>

Definición:

<xs:attribute name="lang" type="xs:string"/>

<xs:attribute name="lang" type="xs:string" default="EN"/>

<xs:attribute name="lang" type="xs:string" fixed="EN"/>

<xs:attribute name="lang" type

Default, fixed, optional/required

XSD - Elementos complejos

- Un elemento complejo es un elemento XML que contiene otros elementos y/o atributos.

- Existen cuatro tipos de elementos complejos:
  - Elementos vacíos
  - Elementos que contienen solamente otros elementos
  - Elementos que contienen solamente texto
  - Elementos que contienen tanto otros elementos como texto

XSD - Tipos de elementos complejos

```
<product pid="1345"/>
```

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

```
<food type="dessert">Ice cream</food>
```

```
<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

XSD - Definición de un elemento complejo

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Definición

XSD - Indicadores

Existen siete tipos de indicadores:

- Indicadores de orden:
  - **All**
  - **Choice**
  - **Sequence**

- Indicadores de occurrencia:
  - **maxOccurs**
  - **minOccurs**

- Indicadores de grupo:
  - **Group name**
  - **attributeGroup name**

XSD - Indicadores de orden: ALL

Especifica que los elementos hijo pueden aparecer en cualquier orden, y que cada elemento hijo puede ocurrir solamente una vez.

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

XSD- Indicadores de orden: CHOICE

Especifica que los elementos hijo puede aparecer (uno o el otro).

```
<xs:element name="person">
 <xs:complexType>
  <xs:choice>
   <xs:element name="employee" type="employee"/>
   <xs:element name="member" type="member"/>
  </xs:choice>
 </xs:complexType>
</xs:element>
```

XSD - Indicadores de orden: SEQUENCE

Especifica que los elementos hijo deben aparecer en estricta secuencia, tal y como se han definido.

```
<xs:element name="person">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="firstname" type="xs:string"/>
     <xs:element name="lastname" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD - Indicadores de ocurrencia

Indicadores maxOccurs y minOccurs (número de veces que un elemento hijo puede ocurrir)

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string  “

       maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

maxOccurs="unbounded"

XSD- Ejercicio

- Escribir XSD

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">

<person>
  <full_name>Hege Refsnes</full_name>
  <child_name>Cecilie</child_name>
</person>

<person>
  <full_name>Tove Refsnes</full_name>
  <child_name>Hege</child_name>
  <child_name>Stale</child_name>
  <child_name>Jim</child_name>
  <child_name>Borge</child_name>
</person>

<person>
  <full_name>Stale Refsnes</full_name>
</person>

</persons>
```

Contenidos

- XML
  - Origen
  - Sintaxis
  - DTDs
  - XML Schema
  - **Procesamiento**

Procesamiento de XML

- Una aplicación que procese documentos XML debe analizarlos. El análisis se realiza usando librerías que implementan los analizadores → **parsers**
- Existen dos tipos de **analizadores sintácticos** XML:
  - Analizadores simples, comprueban que los documentos XML son documentos bien formados.
  - Analizadores de validación, comprueban que los documentos son bien formados y válidos XML de acuerdo a DTDs o esquemas.
- Otra clasificación de los analizadores es la siguiente:
  - Analizadores que tratan el documento como estructura plana (secuencialmente): el programador escribe los procedimientos a ejecutar durante el análisis: etiqueta de apertura, etiqueta de cierre, etc.
  - Analizadores que generan un árbol de análisis. El programador recorre el árbol y consulta las propiedades de los nodos, que representan elementos XML.
- Segundo tipo más práctico, pero más lento. Además, los documentos grandes pueden requerir excesiva memoria para generar el árbol.

Procesamiento de XML

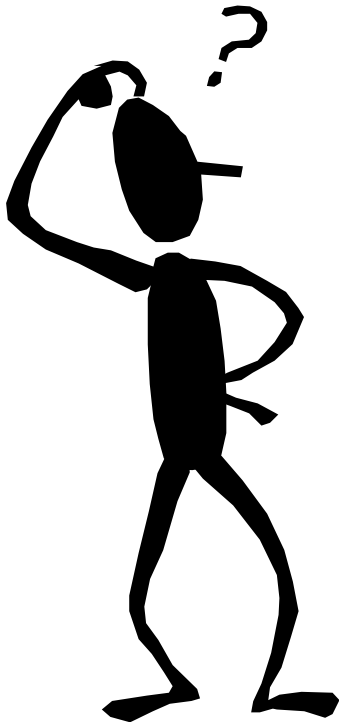
- Características de analizadores
  - Validación de los documentos XML
  - Interfaz (API) para manipular documentos XML
- Dos tipos de parsers
  - **DOM**, script's (java, vb, c++, c#,...) → árbol
  - **SAX**, JAVA → secuencial
    - JRE los tiene integrados

XML - Conclusiones

- XML es un estándar de intercambio de información estructurada entre diferentes plataformas.
- Tecnología sencilla
  - Complementada por otras que lo rodean
- Compatibilidad entre sistemas heterogéneos
- Múltiples lenguajes y formatos se basan en XML
  - **RSS, Atom, SOAP, Office Open XML, LibreOffice**, …
- Diversidad de APIs en diferentes lenguajes y plataformas

Referencias

- M. Morrison (2000) *XML al descubierto*. Prentice-Hall
- B. McLaughlin (2001) *Java & XML*. O'Reilly & Associates
- Extensible Markup Language (XML). http://www.w3.org/XML/
- XML at The Apache Foundation. http://xml.apache.org/
- XML Schema. http://www.w3.org/XML/Schema
- Comparing the Performance of Abstract Syntax Notation One (ASN.1) vs. eXtensible Markup Language (XML). Darren P Mundy , David Chadwick , Andrew Smith



Daniel Sánchez Martínez (danielsm@um.es)
Área de Ingeniería Telemática
Departamento de Ingeniería de la Información y las Comunicaciones (http://www.diic.um.es:8080/diic/)
Facultad de Informática (http://www.um.es/informatica)