XML: eXtensible Markup Languaje

Lenguajes para Internet

Juan Manuel Cueva Lovelle
cueva@lsi.uniovi.es
www.di.uniovi.es/~cueva

Departamento de Informática
Universidad de Oviedo (Asturias, España)

OOTLab www.ootlab.uniovi.es

Lenguajes de Marcas

- Desde el HTML a XHTML
  - HTML
  - XHTML
  - Hojas de Estilo (CSS)
- Desde SGML a XML
  - SGML
  - XML
    - DTD
    - Esquemas XML

Lenguajes de Marcas

HTML (HiperText Markup Language)

AlCapone.html

```
<html>
<head>
<title>Pizzeria Al Capone</title>
</head>
<body bgcolor="blue"  text="yellow"
      link="red"  vlink="white">

<h1>Pizzeria Al Capone</h1>
<p>Lista de enlaces</p>
<ul>
<li><a href="Pizzas.html">
        Tipos de Pizzas</a></li>
<li><a href="http://www.mafia.it">
        Patrocinadores</a></li>
<li><a href="#Contacto">Contacto</a></li>
<ul>

<h2><a name="Contacto">Contacto</a></h2>
<p><font color="red">Dirección:</font>
        C/ Génova Nº 3,  Oviedo, España</p>
<p><font color="red">Teléfono:</font>
        985203040</p>
</body>
</html>
```

Cabecera

Cuerpo

Lista

Enlaces

Detalles de presentación

Página visualizada

Problema:
Presentación y contenido mezclados

Lenguajes de Marcas
Validación XHTML

- XHTML. Es el estándar que contiene la redefinición de HTML en XML.
- Si un sitio web tiene validación de su XHTML mostrará el icono adjunto.
- Para mostrar el icono es necesario incluir el siguiente código XHTML:

```
<p> <a href="http://validator.w3.org/check?uri=referer">
      <img src="http://www.w3.org/Icons/valid-xhtml10"
       alt="Valid XHTML 1.0 Strict"
       height="31"
       width="88"/>
    </a>
</p>
```

- Haciendo clic en el icono comprobará la validez del HTML de la página Web
- Referencias: http://validator.w3.org

Lenguajes de marcas
Hojas de estilo CSS

AlCapone.html

```
<html>
<head>
<title>Pizzeria Al Capone</title>
<link rel="stylesheet" href="pizzeria.css">
</head>
<body>
<h1>Pizzería Al Capone</h1>
<p>Lista de enlaces</p>
<ul>
<li><a href="Pizzas.html">
        Tipos de Pizzas</a></li>
<li><a href="http://www.mafia.it">
        Patrocinadores</a></li>
<li><a href="#Contacto">
        Contacto</a></li>
<ul>


<h2 ><a name="Contacto">Contacto</a></h2>
<p><span class="item">Dirección: </span>
        C/ Génova Nº 3, Oviedo, España</p>
<p><span class="item">Teléfono: </span>
        985203040</p>
</body>
</html>
```

Enlace a hoja de estilo

Sin aspectos visuales

Identificación elementos

pizzeria.css

```
body { color : yellow;
       background: blue
     }
a:link { color: red }
a:visited { color: white }
span.item { color : red }
```

Página visualizada

Lenguajes de Marcas
Hojas de estilo - Reutilización

Misma apariencia

pizzas.html

```
<html>
<head>
<title>Tipos Pizzas</title>
<link rel="stylesheet" href="pizzeria.css">
</head>
<body>
<h1>Pizzas del Restaurante Al Capone</h1>
<table><caption>Tipos de Pizzas</caption>
<th>
<td>Pizza</td><td>Ingredientes</td><td>Precio</td></th>
<tr><td>Barbacoa</td>
    <td>Salsa barbacoa, Mozzarella, Pollo, Bacon, Ternera</td>
    <td>8 &euro;</td></tr>
...
<tr><td>Margarita</td>
    <td>Tomate, Jamón, Queso</td>
    <td>6 &euro;</td>
</tr>
</table>
</body>
</html>
```

Problema:
¿significado de las marcas?
¿Procesamiento automático?

Lenguajes de Marcas
Validación CSS

- Si un sitio web tiene validación de las CSS mostrará el icono adjunto
- Para mostrar el icono es necesario incluir el siguiente código XHTML:

```
<p> <a href="http://jigsaw.w3.org/css-validator/">
      <img style="border:0;width:88px;height:31px“
        src="http://jigsaw.w3.org/css-validator/images/vcss"
        alt ="¡CSS Válido!" />
    </a>

</p>
```

- Haciendo clic en el icono comprobará la validez de las CSS de la página Web

Lenguajes de Marcas: de SGML a XML

- **SGML** (*Standard Generalized Markup Language)*
  - *Utilizado para el intercambio de documentos*
  - *Principio: Separar contenido de la forma de representarlo*
  - *Permite utilizar un conjunto de marcas específico para cada aplicación*
  - *XML es un subconjunto de SGML*
  - *Problema de SGML: Demasiado complicado para su adopción en la Web*
- **XML** (eXtended Markup Language)
  - Desarrollado por el consorcio Web (1995)
  - Versión simplificada de SGML
  - Es un metalenguaje
  - Objetivos:
    - Standard de intercambio de información a través de la Web
    - Formato abierto, independiente de la plataforma
    - Permite utilizar vocabularios específicos de una aplicación
    - Permite la auto-descripción de dichos vocabularios (documentos auto-descritos)
    - Las aplicaciones pueden descubrir el formato de la información y actuar en consecuencia

SGML (Standard Generalized Markup Language)

- Estándar internacional para la definición de la estructura y el contenido de documentos
  - **SGML**: norma ISO 8879:1986(E)
- Es anterior a la Web
- Utiliza un documento tipo DTD (***D****ocument* ***T****ype* ***D****efinition*) que definen su estructura
- La diferencia de SGML con otros lenguajes de marcas es que se pueden utilizar otros caracteres sin ser “<>” para definir las etiquetas, por ejemplo el “-”
- Al ser tan genérico, es difícil de manejar

XML (eXtensible Markup Language)

- Lenguaje de **marcas**
- **Extensible**: Etiquetas no predefinidas
- Diseñado para **describir datos**
- Puede ser estructurado si se usan **DTDs** para validar contenido y sintaxis
  - **Flexibilidad:** se pueden usar o no DTDs
- **Schemas** son similares a los DTD's, pero usan un formato distinto. Son útiles cuando un grupo de documentos XML comparten una serie de reglas
- http://www.w3.org/TR/xml11/

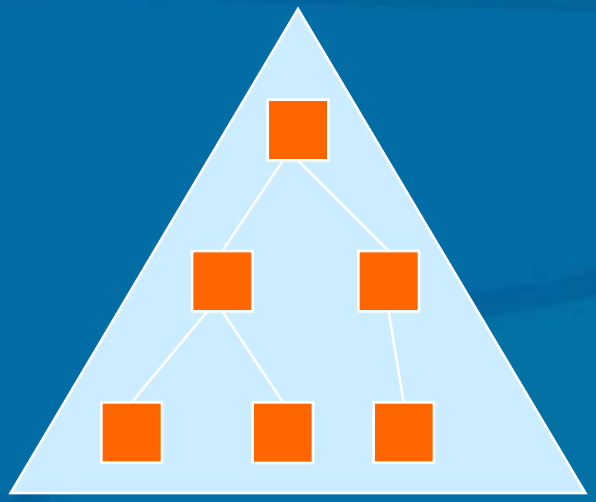
Concepto de documento

- XML especifica el contenido y la estructura, pero no la presentación

```
Esto es
el contenido
del documento
```

CONTENIDO

PRESENTACIÓN

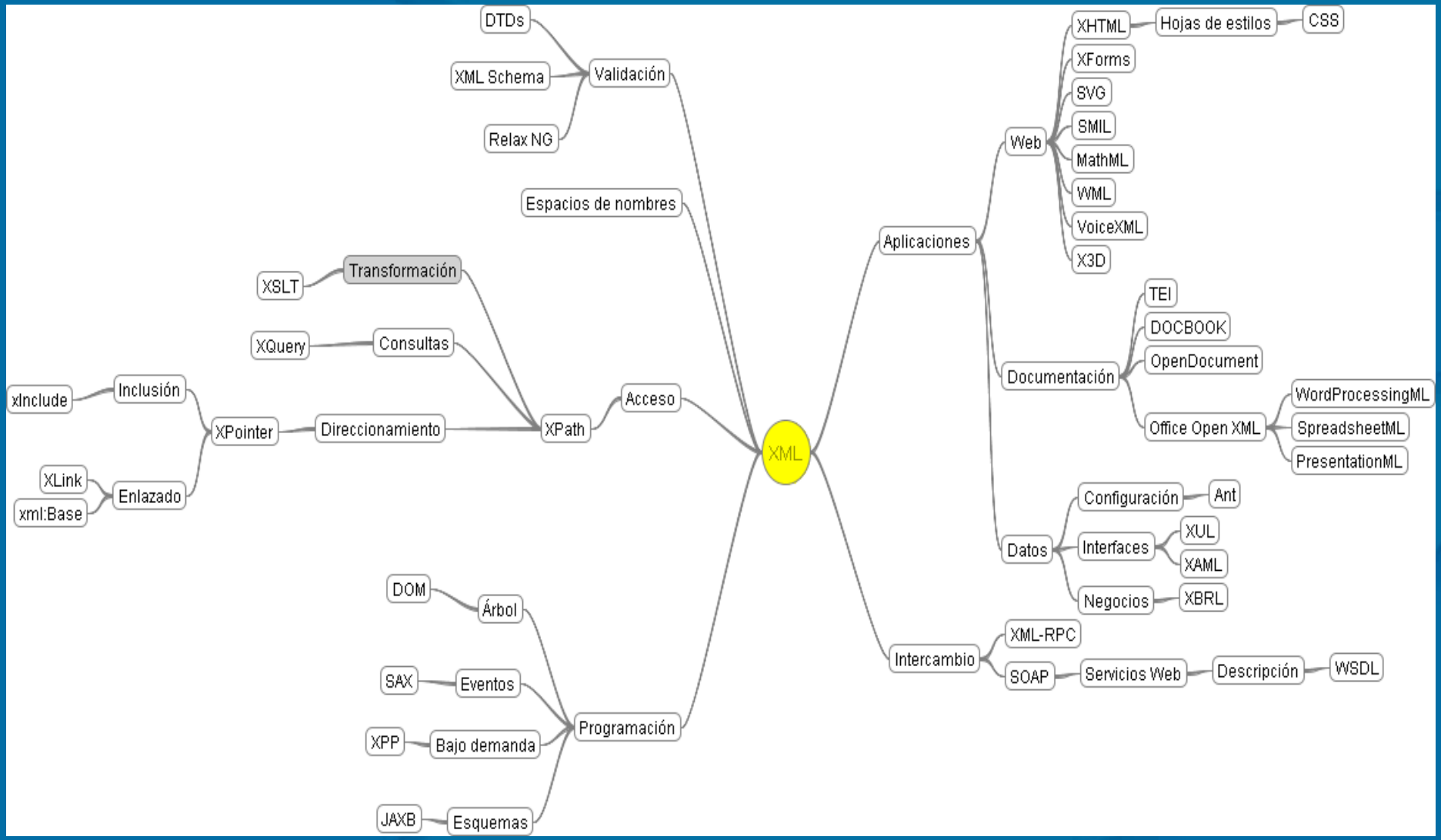


ESTRUCTURA

¿Por qué usar XML?

- Un documento XML puede ser fácilmente procesado y sus datos manipulados
- Existen APIs para procesar esos documentos en Java, C#, C, C++, PHP, ...
- XML define datos portables al igual que Java define código portable

Mapa conceptual XML



Ejemplo de XML

DTD = Declaración de Tipo de Documento

pizzas.xml

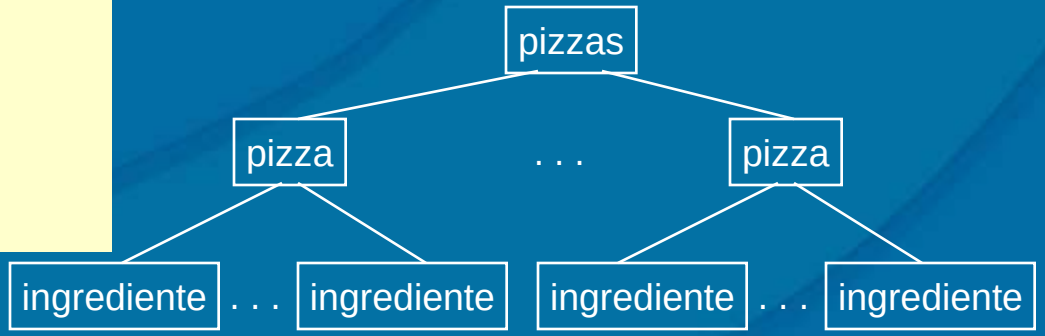
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE pizzas SYSTEM "pizzas.dtd">
<pizzas>
  <pizza nombre="Barbacoa" precio="8">
    <ingrediente nombre="Salsa Barbacoa" />
    <ingrediente nombre="Mozzarella" />
    <ingrediente nombre="Pollo" />
    <ingrediente nombre="Bacon" />
    <ingrediente nombre="Ternera" />
  </pizza>
  . . .
  <pizza nombre="Margarita" precio="6">
    <ingrediente nombre="Tomate" />
    <ingrediente nombre="Jamón" />
    <ingrediente nombre="Queso" />
  </pizza>
</pizzas>
```

Las marcas tienen un significado propio de la aplicación

pizzas.dtd

```
<!ELEMENT pizzas (pizza*)>
<!ELEMENT pizza (ingrediente*)>
<!ELEMENT ingrediente (#PCDATA)>
<!ATTLIST pizza nombre CDATA #REQUIRED>
<!ATTLIST pizza precio CDATA #REQUIRED>
<!ATTLIST ingrediente nombre CDATA #REQUIRED>
```

Estructura de árbol



Más ejemplos XML

alba.xml

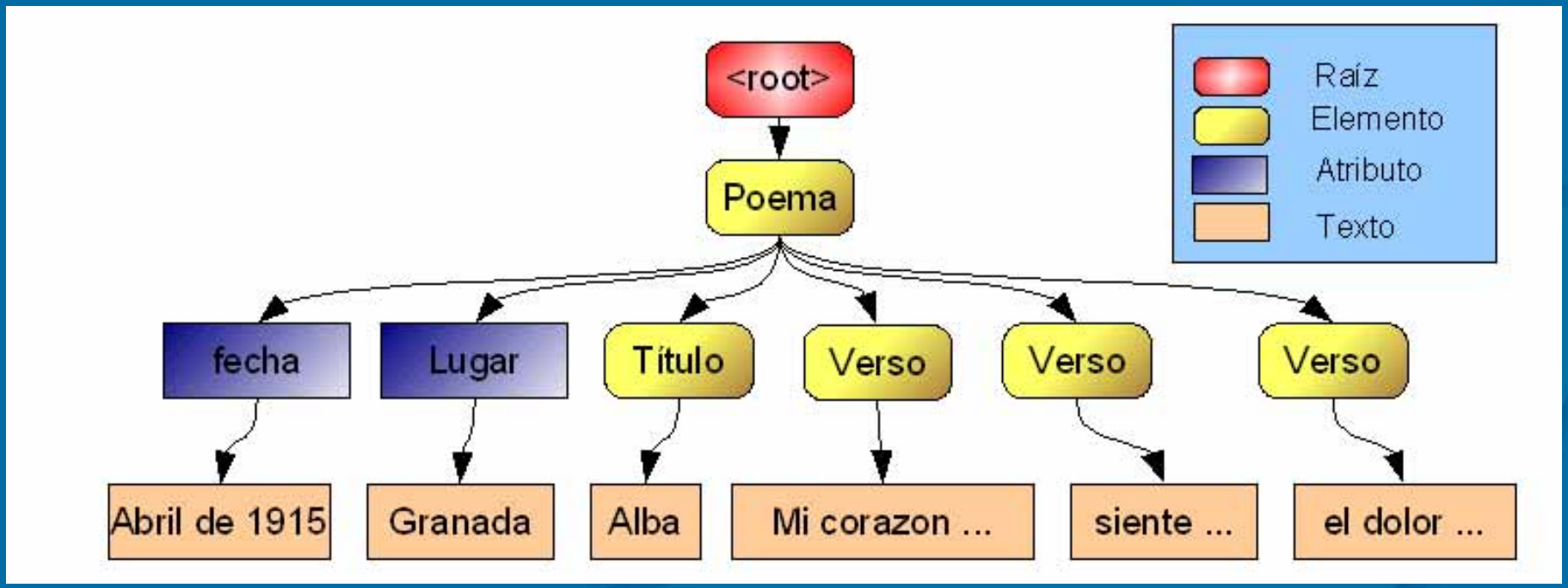
```
<?xml version="1.0" ?>
<!DOCTYPE poema SYSTEM "poema.dtd">
<poema fecha="Abril 1915" lugar="Granada">
<titulo>Alba</titulo>
<verso>Mi corazón oprimido</verso>
<verso>siente junto a la alborada</verso>
<verso>el dolor de sus amores...</verso>
</poema>
```

poema.dtd

```
<!ELEMENT poema ( t i t u l o , v e r s o +)>
<!ELEMENT t i t u l o (#PCDATA)>
<!ELEMENT v e r s o (#PCDATA)>
<!ATTLIST poema
        f echa CDATA #REQUIRED
        l u g a r CDATA #IMPLIED>
```

Opcional

Estructura en árbol



Documento XML

Documento XML = datos + marcado

- *“Información jerarquizada, en forma de texto, que constituye un objeto de datos que puede ser presentado mediante una estructura de árbol, que puede estar almacenado en un único archivo o dividido en varios”*
- Editor:
  - Editor de texto: **Bloc de notas, Notepad++**
  - Editor especializado: **XMLSpy**

Partes de un documento XML

- **Prólogo**
  - Declaración XML
  - Declaración del Tipo de Documento (DTD)
- **Cuerpo**
  - Elementos
  - Atributos
  - Entidades predefinidas
  - Instrucciones de proceso
  - Secciones CDATA
- Comentarios:
  - Se pueden introducir en cualquier lugar del cuerpo o del prólogo, pero nunca dentro de las declaraciones, etiquetas u otros comentarios. Dentro de los comentarios no pueden aparecer los caracteres --

```
<!-- Texto de comentario-->
```

Prólogo: Declaración XML

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

- ***version*** *(obligatoria):* versión de XML usada en el documento
  - 1.0 = versión más habitual
  - 1.1 aumenta capacidad de soporte de Unicode
- ***encoding*** *(opcional)*: la forma en que se ha codificado el documento.
  - Por defecto es UTF-8, aunque podrían ponerse otras, como UTF-16, US-ASCII, ISO-8859-1, etc.
  - UTF-8= caracteres Unicode
  - iso-8859-1 = caracteres latinos
- ***standalone*** (opcional)
  - Indica al procesador XML si un documento es independiente, es decir no usa fuentes externas (standalone=”no”)
  - Indica que se basa en información de fuentes externas, es decir, si depende de declaraciones de marca externas como una DTD externa (standalone=”yes”).

Prólogo: Declaración del Tipo de Documento

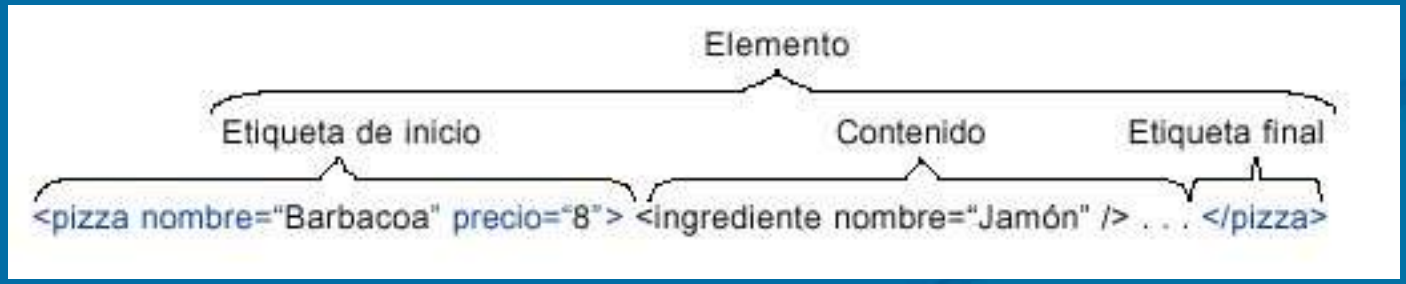
```
<!DOCTYPE nombre SYSTEM|PUBLIC uri>
```

- Declara una DTD que se quiera incorporar al documento
- **DOCTYPE:** indica que la etiqueta contiene una DTD
- **nombre**: declara el nombre de la DTD
- **PUBLIC**: indica que la DTD es pública y está disponible
- **SYSTEM**: La DTD no es pública
- **URI** (*Identificadores únicos de recursos*)
  - URI = URL + URN
    - URL (*Locator*) tiene doble funcionalidad:
      - Identificar recursos
      - Protocolo de acceso
      - Ejemplo: http://www.uniovi.es
    - URN: Nombre único de recursos
      - Ejemplo: urn:isbn:0-395-36341-1

```
<!DOCTYPE pizzas SYSTEM "pizzas.dtd">
```

Cuerpo: Elementos

- El cuerpo del documento está formado por un elemento
- Un elemento está formado por:
  - Una **etiqueta inicial** (nombre entre signos < y >): <etiqueta>
  - La etiqueta inicial puede contener una lista de **atributos** y **valores**:
    <etiqueta atributo="valor">
  - Los contenidos del elemento (puede estar vacío)
  - El elemento debe acabar con una **etiqueta final** con el mismo nombre
  - El **contenido** del elemento es todo lo que hay entre la etiqueta inicial y la final
    - El contenido pueden ser otros elementos



  - En caso de un elemento vacío puede usarse la sintaxis:<etiqueta />

<ingrediente nombre="Jamón" calorías="8"></ingrediente>

| | |

<ingrediente nombre="Jamón" calorías="8" />

- Es necesario cerrar todas las etiquetas
- XML es sensible a mayúsculas/minúsculas

Cuerpo: Elementos (II)

- Sintaxis de los elementos:

  <NombreElemento>Contenido</NombreElemento>

- Representación de elementos (llamados "*singletons*") vacíos:

  <NombreElemento/>

- Anidamiento de elementos jerárquico

```
<Alumno>
   <NombreAlumno>Juan Manuel</NombreAlumno>
   <Edad>18</Edad>
   <Matriculadas>80</Matriculadas>
</Alumno>
```

Cuerpo: Elementos (III)
Convención para nombres

- Consisten de uno o más caracteres sin espacios en el medio.
  - Solo puede comenzar con una letra o con un “_”
  - Los siguientes caracteres pueden ser cualquiera del estándar “Unicode”.
- Sensibles a mayúsculas/minúsculas
- ***Ejemplos:***
  - *Válidos*

    “Juan”
    “_Juan”
    “Juan1234”
    “Juanβ”
  - *No válidos*

    “-Juan”
    “1234”
    “Juan 1234”

Cuerpo: Anidamiento

- Las etiquetas que se abran deben cerrarse sin que se produzcan anidamientos
- *Correcto*

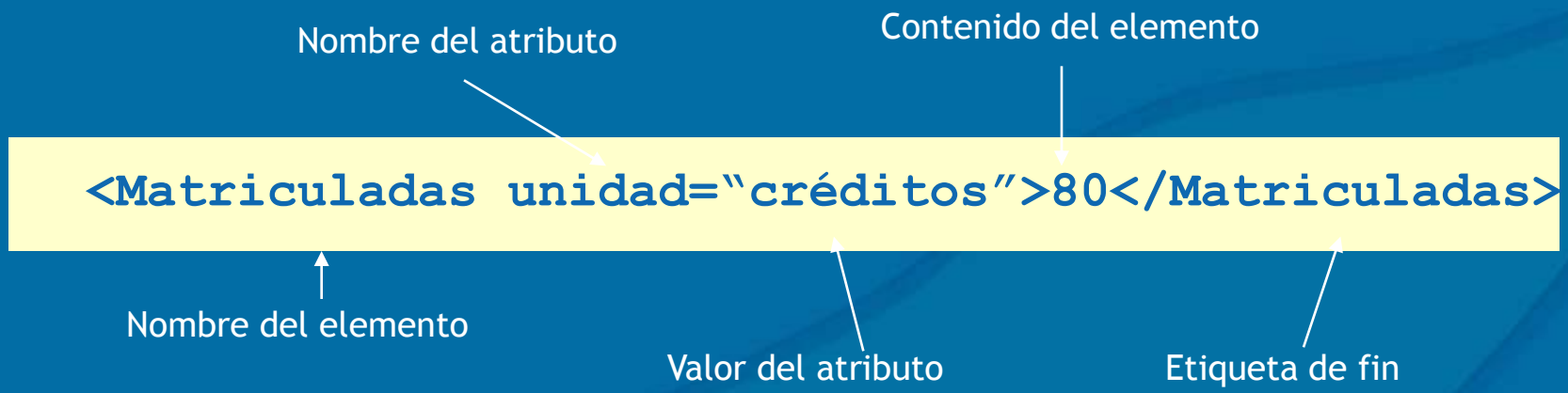
```
<externo>
     <interno> texto </interno>

 </externo>
```

- *Incorrecto*

```
<externo>
   <interno> texto </externo>
</interno>
```

Cuerpo: Atributos

- Los valores de los atributos se escriben entre comillas dobles o simples.
- El orden de atributos no es significativo
- No puede haber nombres de atributos repetidos
- Pueden incluirse valores entrecomillados siempre que las comillas sean diferentes de las externas.
  - Ejemplo: frase="Diego dijo 'digo' "



Cuerpo: Atributos predefinidos

- Hay varios atributos predefinidos.
- Por ejemplo:
  - xml:lang especifica el código del idioma: en (inglés), sp (español), etc.
  - xml:space especifica cómo tratar el espacio en blanco:
    - preserve = mantenerlo
    - default = dejar libertar a la aplicación para tratarlo como quiera
  - xml:base define la URL que identifica las direcciones relativas

Ejemplo de atributos predefinidos

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
 <p xml:lang="en-GB">What colour is it?</p>
 <p xml:lang="en-US">What color is it?</p>
<p xml:lang="sp">¿Qué color es?</p>
 <sp who="Faust" desc='leise' xml:lang="de">
                <l>Habe nun, ach! Philosophie,</l>
                <l>Juristerei, und Medizin</l>
                 <l>und leider auch Theologie</l>
                 <l>durchaus studiert mit heißem Bemüh'n.</l>
 </sp>
```

Cuerpo: Entidades predefinidas

Entidad	Carácter
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

- Cualquier carácter Unicode puede indicarse mediante & seguido del número y acabado por ;

Cuerpo: Instrucciones de proceso

- Mecanismo que permite a los documentos XML contener instrucciones específicas para las aplicaciones que los van a usar, sin que éstas formen parte de los datos del propio documento
- Ejemplos:

- <?xml version='1.0' ?>
- **<?xml-stylesheet type= "text/xsl" href= "MiHojaDeEstilo.xsl"?>**
- **<?php ... ?>**

Cuerpo: Secciones CDATA

- Construcción en XML que permite especificar datos, utilizando cualquier carácter, especial o no, sin que se interprete como marcado XML
- La razón de esta construcción es que a veces es necesario para los autores de documentos XML, poder leerlo fácilmente sin tener que descifrar los códigos de entidades.
- Por ejemplo con código fuente de lenguajes de programación.

Cuerpo: Secciones CDATA (II)

Ejemplo sin CDATA

```
<codigo>
if ( x &lt; 3 &amp;&amp; x &gt; 4)
  printf (&quot; Hola&quot; ) ;
</codigo>
```

Ejemplo con CDATA

```
<codigo>
<! [CDATA[
if ( x < 3 && x > 4)
  printf (” Hola ” ) ;
] ]>
</codigo>
```

Definición de XML bien formado

- **Documento bien formado**
  - Sigue las reglas sintácticas
  - Importante:
    - Contiene un único elemento raíz
    - Todas las etiquetas están correctamente anidadas

```
<pizzas>
  <pizza nombre="Margarita" precio="6">
    <ingrediente nombre="Tomate" />
    <ingrediente nombre="Queso" />
  </pizza>
</pizzas>
```

```
<pizzas>
  <pizza nombre="Margarita" precio="6">
    <ingrediente nombre="Tomate" >
</pizzas>
```

- El documento puede contener varias **instrucciones de procesamiento**
  - Indican cómo debe procesarse el documento

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

Definición de XML válido

- Se puede incluir una declaración del tipo de documento (DTD)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE pizzas SYSTEM "pizzas.dtd">
<pizzas>
  <pizza nombre="Margarita" precio="6">
    <ingrediente nombre="Tomate" />
  </pizza>
</pizzas>
```

pizzas.dtd

```
<!ELEMENT pizzas (pizza*)>
<!ELEMENT pizza (ingrediente*)>
<!ELEMENT ingrediente (#PCDATA)>
<!ATTLIST pizza nombre CDATA #REQUIRED>
<!ATTLIST pizza precio CDATA #REQUIRED>
<!ATTLIST ingrediente nombre CDATA #REQUIRED>
```

- **Documento válido**
  - Está bien formado y
  - La estructura encaja con la declaración del tipo de documento (DTD)

Validación

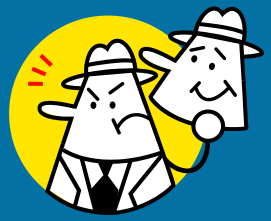
- Es posible validar la estructura de los documentos utilizando diversas alternativas:
  - DTDs
  - XML Schema
    - http://www.w3.org/XML/Schema
  - Relax NG
    - http://relaxng.org/
  - Schematron
    - http://www.schematron.com/

Ventajas de XML

- Es un formato estructurado
- Contiene información y meta-información
  - Ha sido diseñado específicamente para Internet
- Soportado por visualizadores y servidores
  - Numerosas herramientas de procesamiento
  - Legible por personas humanas (fichero de texto)
  - Admite la definición de vocabularios específicos
  - Separa contenido del procesamiento y visualización
  - Aumenta la seguridad mediante la validación de documentos
  - Formato abierto, respaldado por numerosas organizaciones
  - Una vez definido un DTD o un esquema XML común, facilita intercambio de información

Discusión sobre XML: Inconvenientes

- Puede requerir demasiado espacio, ancho de banda y tiempo de procesamiento
  - Documentos largos con mucha información redundante
- Es una sintaxis de documentos, no un lenguaje de programación



```
int main(void) {
 printf("Hola");
 return 0;
}
```

```
<function name="main" type="int">
<arg type="void" />
<call function="printf">
  <param>Hola</param>
</call>
<return value="0"/>
</function>
```

- Es posible crear formatos y vocabularios propietarios
- Puede fomentar la proliferación de vocabularios específicos
- Bueno para texto, malo para datos binarios

```
<?xml version="1.0">
<imagen formato="base64">
DS34JSCDF029876D76523981DFNDF3F2134F5FD019A
FGF23DAND345CD2135911943DCBKAPFGDAJJK32A10
....
</imagen>
```

- Poco eficiente como lenguaje de almacenamiento de bases de datos

DTD (Document Type Definition)

Declaración de Tipo de Documento

DTD (Document Type Definition)

- Conjunto de reglas estándar que deben cumplir un documento XML de un determinado tipo
- Pueden ser vistos como “plantillas” o como gramáticas
- Representados en archivos .dtd
- Determinan:
  - Qué elementos están permitidos en un tipo de documento
  - Relaciones entre los elementos
- Ejemplo: Un artículo de revista puede ser definido así:
    - Título
    - Autores
    - Resumen
    - Palabras clave
    - Nombre de la revista
    - Número de la revista
    - Página inicio-Página final
    - Fecha

Vincular DTDs a documentos XML

- El documento XML que se ajusta a su DTD, se denomina "**valido**" ≠ "**bien-formado**"
- Hay 3 formas de vincular un DTD con un documento XML:
  - Incluyendo el DTD al comienzo del XML con una etiqueta **!DOCTYPE**
    - *Ejemplo:*

```
<!DOCTYPE poema [
<!ELEMENT poema (titulo, verso+)>
. . .
]>
<poema>. . . </poema>
```

  - Colocarlo en un archivo aparte utilizando la misma etiqueta **!DOCTYPE** pero con distinta sintaxis, y así se puede utilizar para validar múltiples documentos XML.
    - *Ejemplo:*

```
<!DOCTYPE factura SYSTEM "factura.dtd">
```

  - Algunas DTDs pueden tener identificadores públicos
    - *Ejemplo:*

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

DTD: Tipos de declaraciones

- ELEMENT
  - Elementos del documento XML
- ATTLIST
  - Lista de atributos de un elemento
- ENTITY
  - Entidad (similares a las macros)
- NOTATION
  - Definen otros tipos de contenidos
  - Facilitan la inclusión de formatos binarios (imágenes, vídeos, sonidos, ...)

DTD: Declaración de Elementos

- Formato:

<!ELEMENT NombreElemento Regla>

- La regla describe el contenido que es almacenado en el elemento:
  - **PCDATA**:el contenido es solo texto, NO tiene anidación dentro de él.
    Ej. `<!ELEMENT enfasis (#PCDATA)>`
  - **ANY**: Puede tener cualquier contenido. Es mejor no usarla y estructurar adecuadamente los documentos
    Ej. `<!ELEMENT batiburrillo ANY>`
  - **EMPTY**: puede ser vacío
    Ej. `<!ELEMENT linea-horizontal EMPTY>`
  - **MIXED:** Puede tener caracteres o una mezcla de caracteres y sub-elementos especificados
    Ej. `<!ELEMENT parrafo (#PCDATA|enfasis)*>`
  - **Element:** Especifica uno o más sub-elementos contenidos en el elemento que se está definiendo
    Ej.`<!ELEMENT html (head, body)>`

Ejemplo DTD

```
<!ELEMENT receta (titulo, ingredientes, procedimiento)>
```

Documento XML válido

```
<receta>
  <titulo>Arroz cocido</titulo>
  <ingredientes>Arroz</ingredientes>
  <procedimiento>Cocer el arroz</procedimiento>
</receta>
```

Documento XML no válido

```
<receta>

  <párrafo>La siguiente receta me la pasó Álvaro</párrafo>
  <titulo>Arroz cocido</titulo>
  <ingredientes>Arroz</ingredientes>
  <procedimiento>Cocer el arroz</procedimiento>
</receta>
```

Error

DTD: Cardinalidad de Elementos

(?) = 0, 1 elemento
(*) = 0 ó más elementos
(+) = 1 ó más elementos
(|) = alternativa
(,) = secuencia
EMPTY = vacío
ANY = cualquier estructura de subelementos
#PCDATA = cadena de caracteres analizados

```
<!ELEMENT  pizza (ingrediente*, inventor?)>
<!ELEMENT servicio (domicilio | restaurante) >
<!ELEMENT ingrediente EMPTY>
<!ELEMENT inventor (#PCDATA)>
```

Ejemplo: libros.xml y libro.dtd

```
<?xml version="1.0">
<libros>
        <libro isbn="9788420633114">
                <titulo>El Aleph</titulo>
                <autor>Jorge Luis Borges</autor>
                <any>1946</any>
                <precio>10</precio>
        </libro>
</libros>
```

- El DTD correspondiente es:

```
<!ELEMENT libros (libro+)>
<!ELEMENT libro  (titulo, autor+, any?, precio)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT any (#PCDATA)>
<!ELEMENT precio (#PCDATA)>
<!ATTLIST libro isbn ID #REQUIRED>
```

DTD: Modelos de contenido

- **Indicador de frecuencia**
  - Siguen directamente a un identificador general, una secuencia o una opción
  - No pueden ir precedidos por espacios en blanco
  - Ejemplo:

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

- **Agrupamientos** utilizando los paréntesis curvos:

  Ejemplo:

```
<!ELEMENT p (font+, (img, br?)*, a*, ul*, ol*)>
```

  el elemento p tiene 1 o más ocurrencias del elemento font, el elemento img seguido por el o los elementos br que halla pueden aparecer 0 o más veces, y así sucesivamente.

- Cuando hay elementos que pueden ser conjuntos **no ordenados**, se representan:

  Ejemplo: **<!ELEMENT p (font | (img, br?) | a | ul | ol) +>**

DTD: Declaración de lista de atributos

- Todos los atributos son declarados como:

```
<!ATTLIST elemento nombre-del-atributo  TYPE Palabra-clave>
```

- Donde palabra clave puede ser:
  - **#REQUIRED**: Es obligatorio especificar el atributo. No tiene valor por defecto.
  - **#IMPLIED**: Se puede omitir el atributo, sin que se adopte automáticamente un valor por defecto
  - **#FIXED**: le da un valor por defecto al atributo

DTD: Atributos

- **Atributos**

#REQUIRED Obligatorio
#IMPLIED Opcional
#FIXED Constante

Tipos de datos

- **CDATA = Cadena de caracteres**
- **NMTOKEN = Palabra (sin espacios)**
- **NMTOKENS = Lista de palabras**
- **Enumeración separada por |**
- **ID = Nombre único (sin duplicados)**
- **IDREF = Su valor debe apuntar a un ID**

```
<!ATTLIST pizza nombre CDATA #REQUIRED>

<!ATTLIST ingrediente nombre CDATA #REQUIRED
                      calorías CDATA #IMPLIED>

<!ATTLIST precio moneda (euros|dólares) #REQUIRED
                valor CDATA #REQUIRED>

<!ATTLIST persona código   ID   #REQUIRED>

<!ATTLIST dueño código IDREF #REQUIRED>

<!ATTLIST impuesto tipo CDATA #FIXED “IVA”>
```

```
<pizza nombre=“4 estaciones” >
  <ingrediente nombre=“Jamón” />
  <precio moneda=“euros” valor=“7” />
</pizza>

<persona código=“23” nombre =“Juan” />
<persona código=“35” nombre =“Pepe” />
<persona código=“37” nombre =“Luis” />

<dueño código=“35” />

<impuesto tipo=“IVA” />
```

DTD: Ejemplo

```
<!ELEMENT mensaje (de, a, texto)>
<!ATTLIST mensaje prioridad (normal | urgente) normal>
<!ELEMENT texto (#PCDATA)>
<!ATTLIST texto idioma CDATA #REQUIRED>
```

```
<mensaje prioridad="urgente">
  <de>Menganito</de>
  <a>Juanita</a>
  <texto idioma="español">
  Lo que le tenga que contar Menganito a Juanita
  </texto>
</mensaje>
```

DTD: Tipos de atributos

- **CDATA** (character data): texto

```
<!ATTLIST mensaje fecha CDATA #REQUIRED>
< mensaje fecha="21 de Mayo de 2003">
```

- **Enumerados**: Sólo pueden contener un valor de entre un número reducido de opciones ("|")

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

- **ID** y **IDREF**
  - El tipo **ID** permite que un atributo determinado tenga un nombre único que podrá ser referenciado por un atributo de otro elemento que sea de tipo **IDREF**
  - Permite implementar un sistema de hipervínculos en un documento XML

```
<!ELEMENT enlace EMPTY>
<!ATTLIST enlace destino IDREF #REQUIRED>
<!ELEMENT capitulo (parrafo)*>
<!ATTLIST capitulo referencia ID #IMPLIED>
```

DTD: Tipos de atributos (II)

- **NMTOKEN Y NMTOKENS:**
  - **Los atributos NMTOKEN (NaMe TOKEN) sólo aceptan los caracteres válidos para nombrar cosas (letras, números, puntos, guiones, subrayados y los dos puntos)**

```
<!ATTLIST mensaje fecha CDATA #REQUIRED>
<mensaje fecha="15 de Diciembre de 1999">

<!ATTLIST mensaje fecha NMTOKEN #REQUIRED>
<mensaje fecha="15-12-1999">
```

- **ENTITY** (permiten definir strings reusables) y ENTITIES

DTD: Declaración de entidades

- La **Entidades** se utilizan para hacer referencia a objetos (ficheros, páginas Web, imágenes,...) que no deben ser analizados sintácticamente según las reglas de XML
- Se declaran mediante "**<!ENTITY**"
- Puede usarse para declarar una abreviatura que se utiliza como una forma más corta de algunos textos
- En otras ocasiones es una referencias a un objeto externo o local

DTD: Tipos de entidades

- Las entidades pueden ser:
  - Internas o Externas
  - Analizadas o No analizadas
  - Generales o Parámetro
- **Entidades generales internas**:

```
<!DOCTYPE texto[
<!ENTITY ovni "Objeto Volador No Identificado">
]>
<texto>
  <titulo> Durmiendo en clase, Álvaro soñó con un &ovni;
  </titulo>
</texto>
```

  - Son básicamente abreviaturas definidas en la sección del DTD del documento XML. Son como macros.
  - Son siempre entidades analizadas

DTD: Tipos de entidades (II)

- **Entidades generales externas analizadas**
  - **Permiten incluir documentos XML externos en el documento actual**

```
<!ENTITY intro SYSTEM "http://www.tecnun.es/intro.xml>
```

- **Entidades no analizadas**

```
<!ENTITY logo SYSTEM "http://www.tecnun.es/logo.gif>
```

- **Entidades parámetro Internas**

```
<!DOCTYPE texto[
<!ENTITY % elemento-alf "!ELEMENT ALF (#PCDATA)>">
%elemento-alf; ]>
```

- **Entidades parámetro Externas**

```
<!DOCTYPE texto[
<!ENTITY % elemento-alf SYSTEM "alf.ent">
...
%elemento-alf; ]>
```

DTD: Espacios de nombres XML

- Permite crear nombres en un documento XML que son identificados por una URI *(**U**niform **R**esource **I**dentifier)*
- Ejemplo de XML valido que introduce la problemática de etiquetas con igual nombre.
- Homonimia: Mismo nombre con diferentes propósitos

```
<país nombre=" Francia">
   <capital>París</capital>
</país>
```

```
<inversión>
  <capital>1200€</capital>
</inversión>
```

¿Cómo combinar en el mismo documento estos vocabularios?

```
<inversiones>
  <país nombre="Francia">
    <capital>París</capital>
    <capital>1200€</capital>
  </país>
  . . .
</inversiones>
```

DTD: Espacios de nombres XML (II)

- Son definidos dentro de los elementos del documento XML. Si un elemento tiene elementos hijos, ellos pueden heredar el *namespace* del padre o sobrescribir el mismo.
- Prefijo para referenciar el *namespace*
  - No puede llamarse ni *xmlns* ni *xml*.
- Declarados dentro de la etiqueta inicial de definición de un elemento
  - Ejemplo:

```
<namespacePrefix:elementName xmlns:namespacePrefix='URL'>
```

DTD: Espacios de nombres XML (III)

- La solución consistirá en asociar a cada etiqueta una URI que identificaría el espacio de nombres al que pertenece
- La URI sirve simplemente para evitar ambigüedad = identificador global único
- Conceptualmente se representaría como:

```
<[http://www.bolsa.com]:inversiones>
<[http://www.geog.es]:pais [http://www.geog.es]:nombre="Francia">
    <[http://www.geog.es]:capital>París</[http://www.geog.es]:capital>
    <[http://www.bolsa.com]:capital>1200€</[http://www.bolsa.com]:capital>
</[http://www.geog.es]:pais>
. . .
</[http://www.bolsa.com]:inversiones>
```

Legibilidad...

DTD: Espacios de nombres XML (IV)

- Para abreviar la sintaxis se asocian alias mediante xmlns:alias="...URI...“
- Todos los elementos en el ámbito de esa declaración que empiecen por alias:nombre pertenecen a dicho espacio de nombres

```
<b:inversiones xmlns:b="http://www.bolsa.com" xmlns:geo="http://www.geo.es">
  <geo:país geo:nombre="Francia">
  <geo:capital>París<geo:capital>
<b:capital>1200€</b:capital>
</geo:pais>
. . .
</b:inversiones>
```

NOTA: Las URIs sólo se utilizan para que el nombre sea único, no son enlaces, ni tienen que contener información

DTD: Espacios de nombres XML (V)

- Asignación dinámica: Es posible ir asociando espacios de nombres a los elementos según van apareciendo

```
<bolsa:inversiones
        xmlns:bolsa="http://www.bolsa.com">
  <geog:país
        xmlns:geog="http://www.geog.es"
        geog:nombre="Francia">
    <geog:capital>París</geog:capital>
    <bolsa:capital>1200€</bolsa:capital>
  </geog:país>
  . . .
</bolsa:inversiones>
```

DTD: Espacios de nombres XML (VI)

- Espacios de nombres por defecto: Mediante **xmlns="..."** se define un espacio de nombres por defecto (sin alias)

```
<inversiones
        xmlns="http://www.bolsa.com">
  <geog:país
        xmlns:geog="http://www.geog.es"
        geog:nombre="Francia">
    <geog:capital>París</geog:capital>
    <capital>1200€</capital>
  </geog:país>
  . . .
</inversiones>
```

Se refiere a
http://www.bolsa.com

DTD: Espacios de nombres XML (VII)

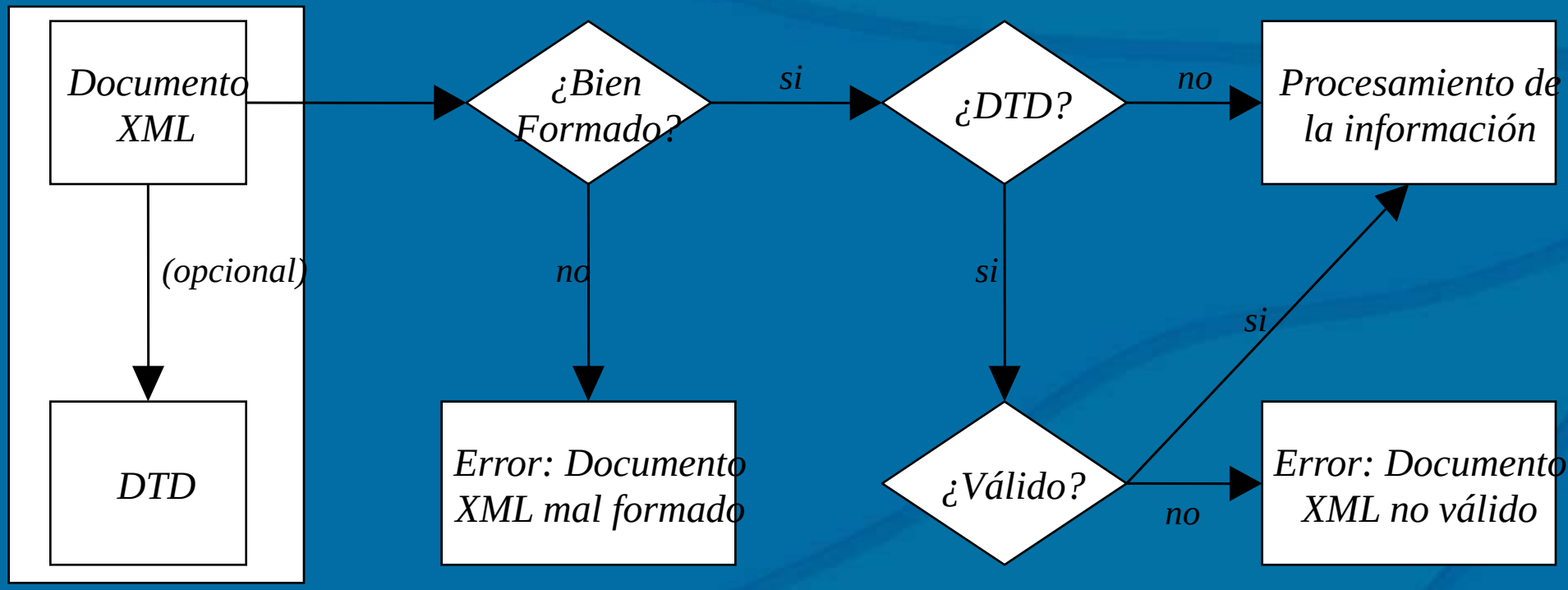
- Incorporación de los espacios de nombres a los DTDs
  - Hay que definir los espacios de nombre usados ya que los DTDs no los contemplan

```
<!DOCTYPE inversiones [
<!ELEMENT inversiones (geog:país*)>
<!ELEMENT geog:país (geog:capital,capital) >
<!ELEMENT geog:capital (#PCDATA)>
<!ELEMENT capital (#PCDATA)>
<!ATTLIST inversiones
        xmlns CDATA #FIXED "http://www.bolsa.com">
<!ATTLIST geog:país
        geog:nombre CDATA #REQUIRED
        xmlns:geog CDATA #FIXED "http://www.geog.es">
]>
```

DTD: Espacios de nombres XML (VIII) Valoración

- Ampliamente utilizados para combinar vocabularios
- Facilitan la incorporación de elementos no previstos inicialmente
- Sintaxis *extraña* al principio
  - Uso de prefijos
  - URIs como elemento diferenciador...pero las URLs también sirven para acceder a recursos
- Complican los DTDs

Proceso de validación documento XML



Esquemas XML

XML Schema

Esquemas XML

- Problemas de los DTDs
  - Difíciles de manipular (no son XML)
  - No son extensibles (una vez definido, no es posible añadir nuevos vocabularios a un DTD)
  - No soportan tipos de datos (ej. enteros, flotantes, etc.)
- XML Schema = Permite definir esquemas de documentos
  - La sintaxis utilizada es XML (La sintaxis de los DTD no era XML!)
  - Soporta la especificación de tipos de datos y tipos definidos por el usuario
  - Soporta comprobación de restricciones numéricas

XML Schema
Lenguajes de Esquemas

- Esquema = definición de estructura de un conjunto de documentos XML
- Validar = Comprobar que un documento sigue un esquema
  - Principal Ventaja: Protección de errores
  - Otras aplicaciones: Edición, comprensión, enlaces de programación, etc.
- Originalmente se utilizaron los DTDs
- Posteriormente a los DTDs se ha desarrollado XML Schema
- Existen Otros:
  - RELAX-NG, Schematron, etc.

XML Schema
Objetivos de Diseño

- Sintaxis XML
- Soporte para Espacios de Nombres
- Mayor expresividad
  - Restricciones numéricas
  - Integridad dependientes del contexto
- Tipos de datos
  - Gran cantidad de tipos de datos predefinidos
  - Creación de tipos de datos por el usuario
- Extensibilidad
  - Inclusión/Redefinición de esquemas
  - Herencia de tipos de datos
- Soporte a Documentación

Características de los XML Schemas

- XML Schemas permiten:
  - Describir estructura
    - Anidación
    - Multiplicidad
    - Ordenamiento
  - Describir tipos
    - Para velocidad operatoria
    - Para mejor almacenamiento
    - Para búsquedas
    - Para ingreso de datos
    - Para detectar errores
- Se almacenan en archivos **.xsd**

Esquemas XML

pizzas.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
<xs:element name="pizzas">
  <xs:complexType>
   <xs:sequence>
     <xs:element ref="pizza" minOccurs='1' maxOccurs='unbounded'/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="pizza">
  <xs:complexType>
   <xs:sequence>
     <xs:element ref="ingrediente" minOccurs='1' maxOccurs='4' />
   </xs:sequence>
   <xs:attribute name="nombre"  type="xs:ID" use='required'/>
   <xs:attribute name="precio"  type="xs:integer" use='required'/>
  </xs:complexType>
 </xs:element>
<xs:element name="ingrediente">
 <xs:complexType>
  <xs:attribute name="nombre" type="xs:string" use='required'
 </xs:complexType>
 </xs:element>
</xs:schema>
```

Elemento raíz schema

Permite especificar rangos de inclusión

Permite especificar tipos

Asociación del fichero XML con el esquema

pizzas.xml

```
<pizzas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance'
        xsi:noNamespaceSchemaLocation='pizzas.xsd'>
. . .
</pizzas>
```

De DTD a Schema: Ejemplo

```
<!ELEMENT address (company?, name, street, city, state, zip, phone+)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

schema en XML

```
<elementType name="address" >
  <sequence>
     <elementType  name="company"    minOccur="0"        maxOccur ="1"/>
     <elementType  name="name"    minOccur="1"           maxOccur ="1"/>
     <elementType  name="street"    minOccur="1"         maxOccur ="1"/>
     <elementType  name="city"      minOccur="1"         maxOccur ="1"/>
     <elementType  name="state"     minOccur="1"         maxOccur ="1"/>
     <elementType  name="zip"       minOccur="1"         maxOccur ="1"/>
     <elementType  name="phone"   minOccur="1"           maxOccur ="5"/>
  <sequence>
</elementType>
```

De DTD a Schema: Reglas

```
*       minOccurrs=0 maxOcurrs=unbounded
+       minOccurrs=1 maxOcurrs=unbounded
?       minOccurrs=0 maxOcurrs=1
,       xs:sequence
|       xs:choice
X       xs:element
```

Elemento raíz Schema

- El espacio de nombres del XMLSchema es usualmente xs:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

- Schema es el elemento raíz del documento

XML Schema: Ejemplo alumnos.xsd

alumnos.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.uniovi.es/alumnos"
           xmlns="http://www.uniovi.es/alumnos">
<xs:element name="alumnos">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="alumno" minOccurs="1" maxOccurs="200"
               type="TipoAlumno"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:complexType name="TipoAlumno">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
    <xs:element name="nacim" type="xs:gYear"/>
  </xs:sequence>
  <xs:attribute name="dni" type="xs:string"/>
 </xs:complexType>
</xs:schema>
```

Elemento raíz schema y espacio de nombres determinado

Permite especificar rangos de inclusión

Permite especificar tipos

XML Schema: Validación alumnos.xml

alumnos.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.uniovi.es/alumnos"
           xmlns="http://www.uniovi.es/alumnos">
<xs:element name="alumnos">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="alumno" minOccurs="1" maxOccurs="200"
               type="TipoAlumno"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:complexType name="TipoAlumno">
  <xs:sequenc
    <xs:eleme
    <xs:eleme
    <xs:eleme
  </xs:sequer
  <xs:attribu
 </xs:complex
</xs:schema>
```

Los espacios de nombres deben coincidir.
También puede usarse:
xsi:noNameSpaceLocation

alumnos.xml

```
<alumnos
  xmlns="http://www.uniovi.es/alumnos"
  xsi:SchemaLocation="http://www.uniovi.es/alumnos
                      alumnos.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
. . .
</alumnos>
```

Vincular un Schema a un documento XML

alumnos.xml

```
<alumnos
  xmlns="http://www.uniovi.es/alumnos"
  xsi:SchemaLocation="http://www.uniovi.es/alumnos
                      alumnos.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
. . .
</alumnos>
```

pizzas.xml

```
<pizzas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation='pizzas.xsd'>
...
</pizzas>
```

XML Schema: Definir un elemento simple

- Sólo puede contener texto (cualquier tipo de dato) pero no otros elementos ni atributos
- Se define como

  <xs:element name="***nombre***" type="***tipo***" **/>**
  donde
  - ***nombre*** es el nombre del elemento
  - Los valores más comunes de tipos de datos (***tipo***) son

    xs:boolean **xs:integer**
    xs:date **xs:string**
    xs:decimal **xs:time**
- Otros atributos:
  - default="***default value***" Valor por defecto de un atributo. Podría definirse otro valor.
  - fixed="***value***" Valor fijo de un atributo. Si no se define, se utiliza ése. Si se define, debe coincidir.
- *Ejemplo:* `<xs:element name="apellido" type="xs:string"/>`

XML Schema: Definir un atributo

- Los atributos se declaran:

  <xs:attribute name="***nombre***" type="***tipo***" />

  Donde:
  - ***nombre*** y ***tipo*** es igual que en xs:element
- Otros atributos:
  - default="***default value***" *si no se especifica otro valor*
  - fixed="***value***“ *no se puede especificar otro valor*
  - use="optional" *el atributo no es obligatorio (por defecto)*
  - use="required" *el atributo debe estar presente*
- *Ejemplo:* `<xs:attribute name="idioma" type="xs:string"/>`

XML Schema: Ejemplos de atributos

```
<xs:complexType name="Círculo">
 <xs:attribute name="radio"
               type="xs:float"
               use="required" />

 <xs:attribute name="color"
               type="Color"
               default="255 0 0"/>

 <xs:attribute name="tipo"
               type="xs:string"
               fixed="jpeg" />
</xs:complexType>
```

Por defecto los atributos son opcionales. Indicar que son obligatorios: `use="required"`

Valor por defecto de un atributo. Podría definirse otro valor.

Valor fijo de un atributo. Si no se define, se utiliza ése. Si se define, debe coincidir.

XML Schema: Definición de tipos

- De los tipos básicos pueden derivarse nuevos tipos, simples o complejos.
  - Los tipos simples contienen texto y se definen mediante **xs:simpleType . Ejemplo:**

```
<xsd:element name="nombre" type="xsd:string"/>
<xsd:attribute name="identificación" type="xsd:ID"/>
```

  - y los complejos pueden contener cualquier combinación de contenido de elementos , información de caracteres y atributos y se definen con **xs:complexType**

XML Schema: Tipos Simples

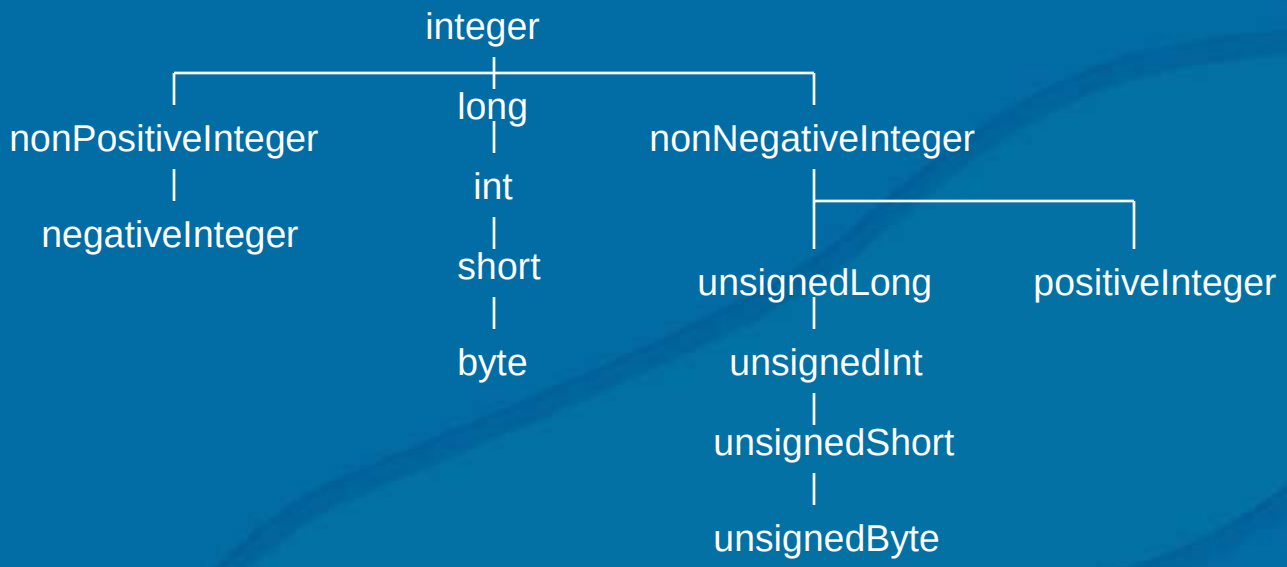
- No pueden contener elementos o atributos
- Pueden ser:
  - Predefinidos o *built-in* (Definidos en la especificación)
    - Primitivos
    - Derivados
  - Definidos por el usuario (a partir de tipos predefinidos)

XML Schema: Tipos Primitivos

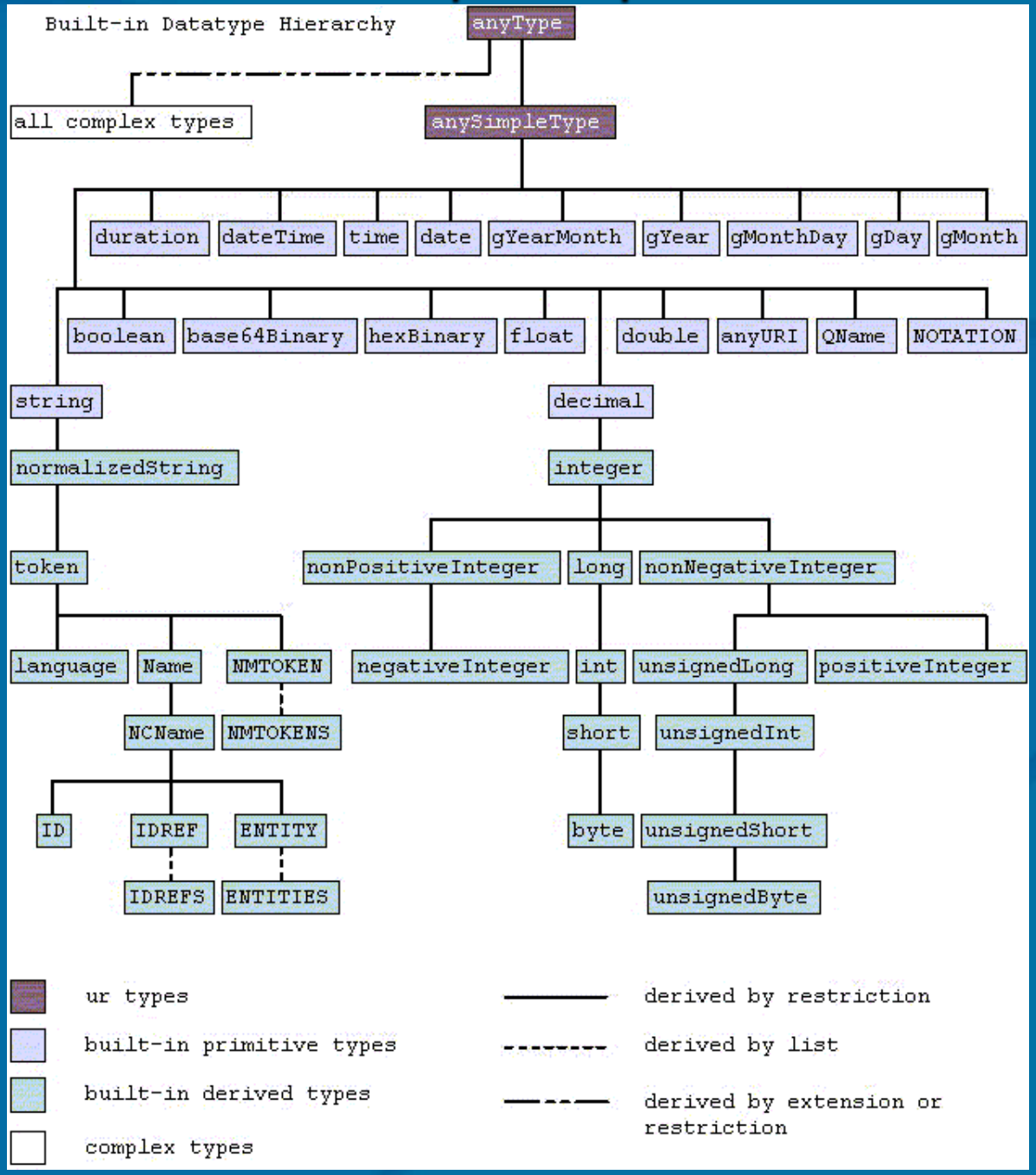
- string
- boolean
- number, float, double
- duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth
- hexBinary, base64Binary
- anyURI
- QName = Nombre cualificado con espacio de nombres
- NOTATION = Notación binaria (similar a DTD)

XML Schema: Tipos Derivados

- normalizedString, token, language
- IDREFS, ENTITIES, NMTOKEN, NMTOKENS, Name, NCName, ID, IDREF, ENTITY
- integer, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, positiveInteger



XML Schema: Jerarquía de tipos



XML Schema: Facetas de Tipos

- Facetas fundamentales:
  - *equal:* Igualdad entre valores de un tipo de datos
  - *ordered:* Relaciones de orden entre valores
  - *bounded:* Límites inferiores y superiores para valores
  - *cardinality:* Define si es finito o infinito (numerable, no numerable)
  - *numeric:* Define si es numérico o no
- Facetas de restricción
  - *length, minlength, maxlength:* Longitud del tipo de datos
  - *pattern:* Restricciones sobre valores mediante expresiones regulares
  - *enumeration:* Restringe a una determinada enumeración de valores
  - *whitespace:* Define política de tratamiento de espacios (preserve/replace, collapse)
  - *(max/min)(in/ex)clusive:* Límites superiores/inferiores del tipo de datos
  - *totaldigits, fractionDigits:* número de dígitos totales y decimales

XML Schema: Restricciones

- Restricciones: Permiten restringir el valor que se le puede dar a un elemento o atributo XML.
- Tipos:
  - Sobre valores
  - Sobre un conjunto de valores
  - Sobre series de valores
  - Sobre espacios en blanco

XML Schema: Restricciones sobre valores

- La forma general de establecer una restricción sobre un valor es:

```
<xs:element  name="name">     (o xs:attribute)
      <xs:restriction base="type">
           ... Las restricciones ...
      </xs:restriction>
  </xs:element>
```

- Por ejemplo:

```
<xs:element  name="edad">
        <xs:restriction base="xs:integer">
             <xs:minInclusive value="0">
             <xs:maxInclusive value="140">
        </xs:restriction>
</xs:element>
```

Ejemplo

Restricciones sobre valores

```
<xs:simpleType name="mes">
<xs:restriction base="xs:integer">
  <xs:minInclusive value="1" />
  <xs:maxInclusive value="31" />
</xs:restriction>
</xs:simpleType>
```

Restricciones sobre un conjunto de valores: Enumeration

- Restringe el valor a un conjunto de valores
- Ejemplo:

```
<xs:element name=“estación">
      <xs:simpleType>
          <xs:restriction  base="xs:string">
              <xs:enumeration value=“primavera"/>
              <xs:enumeration value=“verano"/>
              <xs:enumeration value=“otoño"/>
              <xs:enumeration value=“invierno"/>
          </xs:restriction>
      </xs:simpleType>
  </xs:element>
```

Ejemplo enumeración

```
<xs:simpleType name="TipoCarrera">
<xs:restriction base="xs:token">
 <xs:enumeration value="Gestión"/>
 <xs:enumeration value="Sistemas"/>
</xs:restriction>
</xs:simpleType>
```

Restricciones sobre series de valores

- Para limitar el contenido del elemento XML definiendo las series de números y letras que se pueden usar usaremos patrones (expresiones regulares):

```
<xs:element name="letra">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-z]"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

XML Schema: Expresiones regulares

```
<xs:simpleType name="NIF">
 <xs:restriction base="xs:token">
   <xs:pattern value="\d{7,8}[A-Z]" />
 </xs:restriction>
</xs:simpleType>

<xs:element name="nif" type="NIF" />
```

```
<nif>9394173J</nif>
```

```
<nif>11079845M</nif>
```

Ejemplos de expresiones regulares

EXPRESIÓN REGULAR	POSIBLES VALORES
Elemento \d	Elemento 2
a*b	b, ab, aab, aaab, …
[xyz]b	xb, yb, zb
a?b	b, ab
a+b	ab, aab, aaab, …
[a-c]x	ax, bx, cx

Ejemplos de expresiones regulares (II)

`[a-c]x`	ax, bx, cx
`[^0-9]x`	Carácter ≠ dígito seguido de x
`\Dx`	Carácter ≠ dígito seguido de x
`(pa){2}rucha`	paparucha
`.abc`	Cualquier carácter seguido de abc
`(a|b)+x`	ax, bx, aax, bbx, abx, bax,...
`a{1,3}x`	ax, aax, aaax
`\n`	Salto de línea
`\p{Lu}`	Letra mayúscula
`\p{Sc}`	Símbolo de moneda

Restricciones sobre espacios en blanco

- whiteSpace indica lo que hay que hacer con los espacios en blanco
  - value="preserve“ Mantiene los espacios en blanco como están
  - value="replace" Cambia los espacios en blanco (saltos, tab…) por espacios
  - value="collapse" Reemplaza todas las secuencias de espacios en blanco por un sólo espacio en blanco

Restricciones en números

- minInclusive: El número debe ser ≥ ***value***
- minExclusive: El número debe ser > ***value***
- maxInclusive: El número debe ser ≤ ***value***
- maxExclusive: El número debe ser < ***value***
- totalDigits: El número debe tener exactamente ***value*** dígitos
- fractionDigits: El número no debe tener más de ***value*** dígitos después del punto decimal.

Restricciones en strings

- length – el string debe contener exactamente ***value*** caracteres
- minLength – el string debe contener al menos ***value*** caracteres
- maxLength – el string no debe contener más de ***value*** caracteres
- pattern – el ***value*** es una expresión regular que el string debe cumplir

XML Schema: Listas

- Se pueden aplicar las facetas: length, maxLength, minLength, enumeration

```
<xs:simpleType name="ComponentesRGB">
 <xs:list itemType="ComponenteRGB"/>
</xs:simpleType>

<xs:simpleType name="ComponenteRGB">
<xs:restriction base="xs:nonNegativeInteger">
   <xs:maxInclusive value="255" />
</xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="ColorRGB">
   <xs:restriction base="ComponentesRGB">
     <xs:length value="3" />
   </xs:restriction>
 </xs:simpleType>
```

```
<color>255 255 0</color>
```

XML Schema: Uniones

```
<xs:simpleType name="TipoNota">
<xs:union>
  <xs:simpleType>
    <xs:restriction base="xs:float">
      <xs:maxInclusive value="10" />
      <xs:minInclusive value="0" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="No presentado" />
    </xs:restriction>
  </xs:simpleType>
</xs:union>
</xs:simpleType>

<xs:element name="nota" type="TipoNota" />
```

```
<nota> 5.75 </nota>
```

```
<nota> No presentado </nota>
```

XML Schema: Elementos complejos

- Elementos que contienen otros elementos hijo o que tienen atributos.
- Se suelen dividir en 4 tipos:
  - Elementos vacíos
  - Elementos no vacíos con atributos
  - Elementos con elementos hijos
  - Elementos con elementos hijos y con “texto” o valor propio

XML Schema: Sintaxis de Elementos complejos

- Se definen como:

```
<xs:element   name="nombre">
    <xs:complexType>
        ...
    </xs:complexType>
</xs:element>
```

- Ejemplo:

```
<xsd:element name="empleado">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nombre" type="xsd:string"/>
      <xsd:element name="apellidos" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

XML Schema: Elementos complejos Indicadores

- De Orden:
  - En un determinado orden o secuencia: **sequence**
  - Pudiendo el autor del documento escoger alguno de los elementos del grupo: **choice**
  - Dar al autor total libertad tanto respecto al orden como a la selección de elementos: **all**
- De ocurrencias: cardinalidades

XML Schema: Tipos Complejos-Secuencia xs:sequence

- *Secuencia: Construcción básica mediante enumeración de elementos*
  - Elementos que contienen elementos

```
<xs:complexType name="TipoAlumno">
  <xs:sequence>
    <xs:element name="nombre"    type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
    <xs:element name="nacim" type="xs:gYear"
                minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="dni" type="xs:integer"/>
 </xs:complexType>
```

```
<alumno dni="9399390">
 <nombre>Juan</nombre>
 <apellidos>García García</apellidos>
 <nacim>1985</nacim>
</alumno>
```

XML Schema: Tipos Complejos-Alternativa

xs:choice

- *Choice : Representa alternativas*
  - *¡ Atención ! : Es una o-exclusiva*

```
<xs:complexType name="Transporte">
   <xs:choice>
    <xs:element name="coche" type="xs:string"/>
     <xs:element name="tren" type="xs:string"/>
     <xs:element name="avión" type="xs:string"/>
   </xs:choice>
 </xs:complexType>
```

```
<transporte>
<coche>Renault R23</coche>
</transporte>
```

XML Schema: Tipos Complejos-Secuencias no ordenadas

xs:all

- *all = Todos los elementos en cualquier orden*
- *En los DTDs se requería enumerar las combinaciones:*
  (A,B,C)|(A,C,B)|...|(C,B,A)

```
<xs:complexType name="TipoLibro">
  <xs:all>
      <xs:element name="autor" type="xs:string"/>
      <xs:element name="título" type="xs:string"/>
  </xs:all>
</xs:complexType>
<xs:element name="libro" type="TipoLibro" />
```

```
<libro>
 <autor>Juanita la Loca</autor>
 <título>No estoy loca</título>
</libro>
```

```
<libro>
  <título>El Quijote</título>
  <autor>Cervantes</autor>
 </libro>
```

XML Schema: Tipos complejos - Cardinalidades

- minOccurs y maxOccurs
- Se utilizan para indicar el número máximo y mínimo de veces que puede aparecer un elemento hijo de un elemento complejo
  - minOccurs= 0 : Opcionalidad
  - maxOccurs = unbounded: no existe valor límite

Ejercicio

- Transforma el modelo de contenido

$$(a | (b , c?))^*$$

en una definición mediante XMLSchema

Solución

```
<xs:complexType>
  <xs:choice minOccurs='0'
  maxOccurs='unbounded'>
      <xs:element ref='a'/>
      <xs:sequence>
            <xs:element ref='b'/>
            <xs:element ref='c'
            minOccurs='0' maxOccurs='1'/>
      </xs:sequence>
  </xs:choice>
</xs:complexType>
```

XML Schema: Tipos Complejos-Contenido Mixto

- *El contenido Mixto permite mezclar texto con elementos*
- Se añade mixed="true" al elemento xs:complexType
- El texto no se menciona en el elemento y puede ir en cualquier sitio (básicamente se ignora)

```
 <xs:complexType name="TCom" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="emph" type="xs:string"/>
  </xs:choice>
</xs:complexType>

<xs:element name="comentarios" type="TCom" />
```

```
<comentarios>
 Es un poco <emph>listillo</emph>
</comentarios>
```

XML Schema: Agrupaciones

- Es posible nombrar agrupaciones de elementos y de atributos para hacer referencias a ellas

```
<xs:group name="nombApell">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:group>
```

```
<xs:complexType name="TipoAlumno">
 <xs:group ref="nombApell" />
 <xs:element name="carrera" type="xs:string"/>
</xs:complexType>
```

XML Schema
Tipos *Anónimos* versus *Con nombre*

```
<xs:element name="alumno">
<xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:element>
```

Anónimo

+ legible

```
…
<xs:element name="alumno" type="TipoAlumno"/>
…
<xs:ComplexType name="TipoAlumno">
<xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:ComplexType>
```

Con nombre

+ Reutilizable

XML Schema
Otra posibilidad: Referencias

```
<xs:element name="alumno">
<xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:element>
```

```
<xs:element name="alumnos">
<xs:sequence>
  <xs:element ref="alumno" />
</xs:sequence>
</xs:element>
```

XML Schema: Tipos Derivados por Extensión

- Similar a las subclases de POO. Consiste en heredar elementos de un tipo base

```
<xs:complexType name="Figura" >
  <xs:attribute name="color" type="Color"/>
 </xs:complexType>

<xs:complexType name="Rectángulo">
 <xs:complexContent>
 <xs:extension base="Figura">
 <xs:attribute name="base" type="xs:float" />
 <xs:attribute name="altura" type="xs:float" />
 </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="Círculo">
...similar pero incluyendo el radio
</xs:complexType>
```

XML Schema: Tipos Derivados por Extensión (II)

- Los tipos derivados pueden utilizarse en los mismos sitios que la clase base

```
<xs:element name="figuras">
   <xs:complexType>
     <xs:sequence>
      <xs:element name="figura" type="Figura"
                  maxOccurs="unbounded" />
     </xs:sequence>
   </xs:complexType>
  </xs:element>
```

```
<figuras>
<figura base="23" altura="3" xsi:type="Rectángulo" />
<figura radio="3" xsi:type="Círculo" />
</figuras>
```

Es necesario especificar el tipo mediante `xsi:type`

XML Schema: Tipos Abstractos

- Al igual que en la POO se pueden declarar tipos abstractos
- Mediante abstract="true" se declara un tipo como abstracto.
  - Ese tipo no puede usarse directamente
- También es posible limitar la derivación de tipos
  final="restriction"

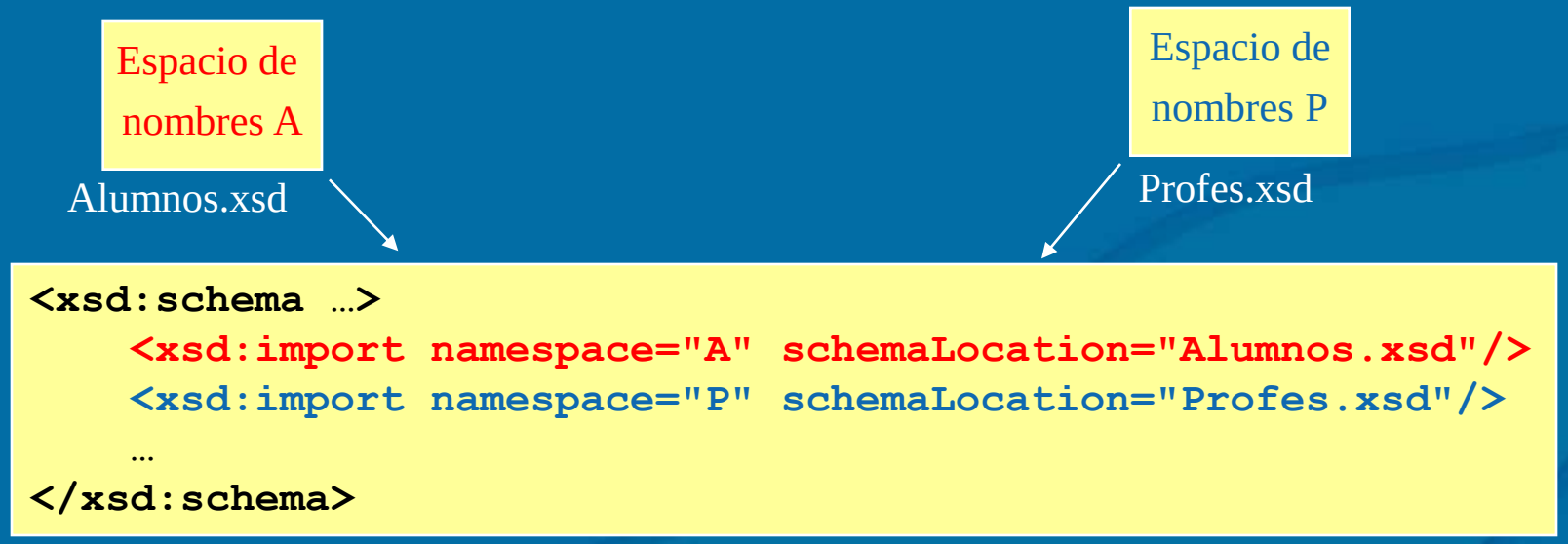
```
<xs:complexType name="Figura" abstract="true">
  <xs:attribute name="color" type="Color"/>
 </xs:complexType>
```

XML Schema: Inclusión de Esquemas

- *include* permite incluir elementos de otros esquemas
  - Los elementos deben estar en el mismo espacio de nombres
  - Es como si se hubiesen tecleado todos en un mismo archivo

XML Schema: Importación de Esquemas

- *import* permite incluir elementos de otros esquemas con distintos espacios de nombres



XML Schema: Redefinición de Esquemas

- *redefine* es similar a *include* pero permite modificar los elementos incluidos.

Añade el elemento `nota`

```
<xs:redefine schemaLocation="Alumnos.xsd">
  <xs:complexType name="TipoAlumno">
    <xs:complexContent>
      <xs:extension base="TipoAlumno">
         <xs:sequence>
           <xs:element name="nota" type="Nota" />
         </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>
```

AlumnosConNotas.xsd

XML Schema: Claves y Unicidad

- Los DTDs proporcionaban el atributo ID para marcar la unicidad (un valor ID era único en todo el documento)
- XML Schema tiene más posibilidades:
  - Indicar que un elemento es único (**unique**)
  - Definir atributos únicos
  - Definir combinaciones de elementos y atributos como únicos
  - Distinción entre unicidad y claves (**key**)
    - Clave = además de ser único, debe existir y no puede ser nulo.
  - Declarar el rango de un documento en el que algo es único

XML Schema: Claves y Unicidad (II)

```
<xs:complexType name="Alumnos">
 <xs:sequence>
  <xs:element name="Alumno" type="TipoAlumno"/>
 </xs:sequence>
  <xs:key name="DNI">
    <xs:selector xpath="a:alumno"/>
    <xs:field xpath="a:dni"/>
  </xs:key>
</xs:complexType>
```

Es necesario incluir el espacio de nombres (XPath)

La clave puede formarse para atributos y elementos

```
<xs:key name="DNI">
    <xs:selector xpath="a:alumno"/>
    <xs:field xpath="a:nombre"/>
    <xs:field xpath="a:apellidos"/>
</xs:key>
```

Una clave puede estar formada por varios elementos

XML Schema: Claves y Unicidad (III)

```
<xs:complexType name="Alumnos">
 <xs:sequence>
  <xs:element name="Alumno" type="TipoAlumno"/>
 </xs:sequence>
  <xs:unique name="DNI">
    <xs:selector xpath="a:alumno"/>
    <xs:field xpath="a:dni"/>
  </xs:unique>
</xs:complexType>
```

Unique especifica que debe ser único, pero podría no existir

XML Schema: Referencias a Claves

- keyref especifica que debe hacer referencia a una clave (Claves Externas)

```
<xs:element name="clase">
 <xs:sequence>
  <xs:element name="alumnos" ...
  <xs:element name="delegado" ...
 </xs:sequence>

<xs:key name="DNI">
 <xs:selector xpath="a:alumnos/a:alumno"/>
 <xs:field xpath="a:dni"/>
</xs:key>

<xs:keyref name="Delegado" refer="DNI">
  <xs:selector xpath="a:delegado"/>
  <xs:field xpath="a:dni"/>
</xs:keyref>
```

XML Schema: Valores Nulos

- Indicar que un elemento puede ser nulo sin estar vacío.
  - Vacío (Empty): Un elemento sin contenido
  - Nulo (Nil): Un elemento que indica que no hay valor

```
<xsd:element name="Persona">
  <xsd:complexType>
  <xsd:element name="nombre" type="xsd:NMTOKEN"/>
  <xsd:element name="primerApell" type="xsd:NMTOKEN"/>
  <xsd:element name="segundoApell" type="xsd:NMTOKEN"
                                   nillable="true"/>
 </xsd:complexType>
</xsd:element>
```

```
<persona>
      <nombre>John</nombre>
      <primerApell>Smith</primerApell>
      <segundoApell xsi:nil="true"/>
</persona>
```

El segundo apellido puede ser un **NMTOKEN** o estar indefinido

XML Schema: Incluir cualquier contenido

- any indica cualquier contenido de un determinado espacio de nombres
- anyAttribute cualquier atributo de un espacio de nombres

```
<xs:complexType name="Comentario">
 <xs:sequence>
  <xs:any namespace="http://www.w3.org/1999/xhtml"
          minOccurs="1"
          processContents="skip" />
 </xs:sequence>
 <xs:anyAttribute
      namespace="http://www.w3.org/1999/xhtml"/>
</xs:complexType>
```

También puede declararse
##any, ##local, ##other

```
<comentarios>
  <html:p>Es un
     <html:emph>Listillo</html:emph>
  </html:p>
</comentarios>
```

Otros valores
strict = obliga a validar
lax = valida si es posible

XML Schema: Generalización de comentarios

- *annotation*: mecanismo para documentar los componentes de un esquema, haciendo una función similar a los comentarios
- *appinfo:* proporciona información a otras aplicaciones, lo que supone un procesamiento externo al propio documento
- *documentation:* texto o referencia a texto dentro del elemento *annotation*

Ejemplo

```
<xsd:complexType nombre="elemento">
  <xsd:annotation">
     <xsd:documentation>
       este elemento es ..
     </xsd:documentation>
  </xsd:annotation">
</xsd:complexType>
```

XML Schema: Limitaciones

- No soporta entidades. Mecanismo para crear macros

  <!ENTITY &texto; "Esto texto se repite muchas veces" >

  - Es necesario seguir usando los DTDs ☹
- Lenguaje de Restricciones limitado
  - Ejemplo: ¿Verificar valor total = suma de valores parciales?
- Sensibilidad al contexto limitada
  - Por ejemplo: Especificar que el contenido depende del valor de un atributo

    <transporte tipo="coche"> ...</transporte>

    <transporte tipo="avión"> ...</transporte>
- Tamaño de archivos XML Schema puede ser excesivo
- Legibilidad de las especificaciones...XML no siempre es legible
- Complejidad de la especificación:
  - Muchas situaciones/combinaciones excepcionales
- Otras propuestas: Relax-NG, Schematron, etc.

Ejercicios

- Creación ficheros XML y validación mediante Esquemas
- Herramienta
  - Validador de XML Schemas:

    http://www.w3.org/2001/03/webdata/xsv

Referencias

- Recomendación XML Schemas:
  - Introducción a los conceptos: *http://www.w3.org/TR/xmlschema-0/*
  - Estructuras: *http://www.w3.org/TR/xmlschema-1/*
  - Tipos de datos: *http://www.w3.org/TR/xmlschema-2/.*
- Validador de XML Schemas:

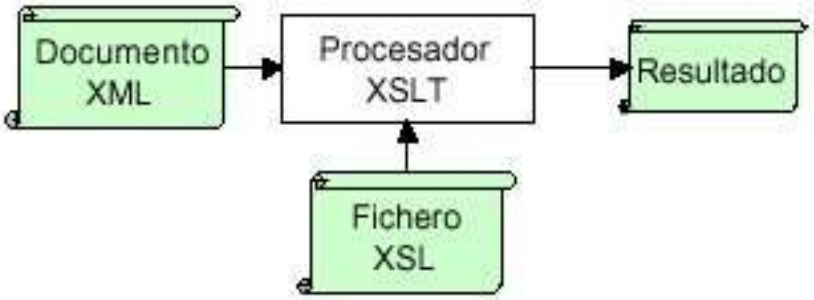
  http://www.w3.org/2001/03/webdata/xsv

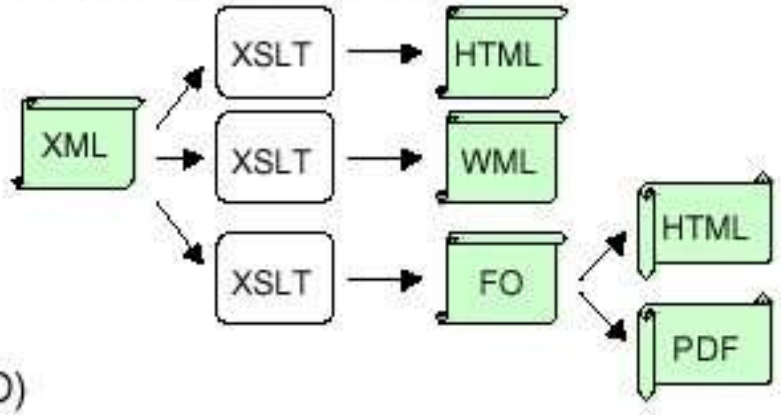
XSLT

XML **S**tylesheets **L**anguage for **T**ransformation

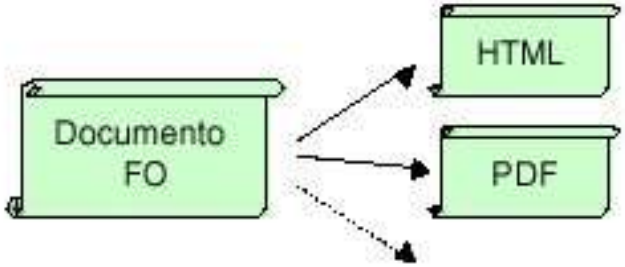
Transformación de documentos XML

- XSL (eXtensible Stylesheet Language)
  - XSL Transformations (XSLT)
    - Lenguaje de transformación de documentos
    - Utiliza XPath para hacer referencias a partes de un documento





  - XSL-Formatting Objects (XSL-FO)
    - Lenguaje que incluye instrucciones de formateo independientes del dispositivo



El problema de presentar un documento XML

- **XML no incorpora ninguna semántica intrínseca de presentación**
- Soluciones
  - **XML+CSS**
  - **XML+XSL (Extensible Stylesheet Language, "lenguaje extensible de hojas de estilo").**
    - Familia de lenguajes basados en XML
    - Permite describir cómo la información contenida en un documento XML debe ser transformada o formateada para su presentación en un medio específico

¿Por qué XSL?

- Está diseñado para integrarse en la arquitectura XML.
- Es mucho más potente que CSS.
  - CSS no tiene capacidades de transformación.
- Cada vez hay más herramientas para XSL.

CSS versus XSL

	CSS	XSL
¿Puede ser usada con HTML?	SI	NO
¿Puede ser usado con XML?	SI	SI
¿Lenguajes de transformación?	NO	SI
¿Sintaxis?	CSS	XML

XSL

- Formado por:
  - XSLT (Extensible Stylesheet Language Transformations): permite convertir documentos XML de una sintaxis a otra (por ejemplo, de un XML a otro o a un documento HTML).
  - XSL-FO (lenguaje de hojas extensibles de formateo de objetos): permite especificar el formato visual con el cual se quiere presentar un documento XML, es usado principalmente para generar documentos PDF.
  - XPath, o XML Path Language: sintaxis (no basada en XML) para acceder o referirse a porciones de un documento XML.

Estas tres especificaciones son recomendaciones oficiales del W3C.

Transformación XSLT

- Transformación estructural:
  - Los datos son convertidos de la estrucutra de entrada (XML) a una estructura que refleje la salida deseada
- Formato (pdf, html, etc.)
  - La salida de datos, generados por la nueva estructura se entrega en el formato requerido (PDF, HTML, delimitado por comas)

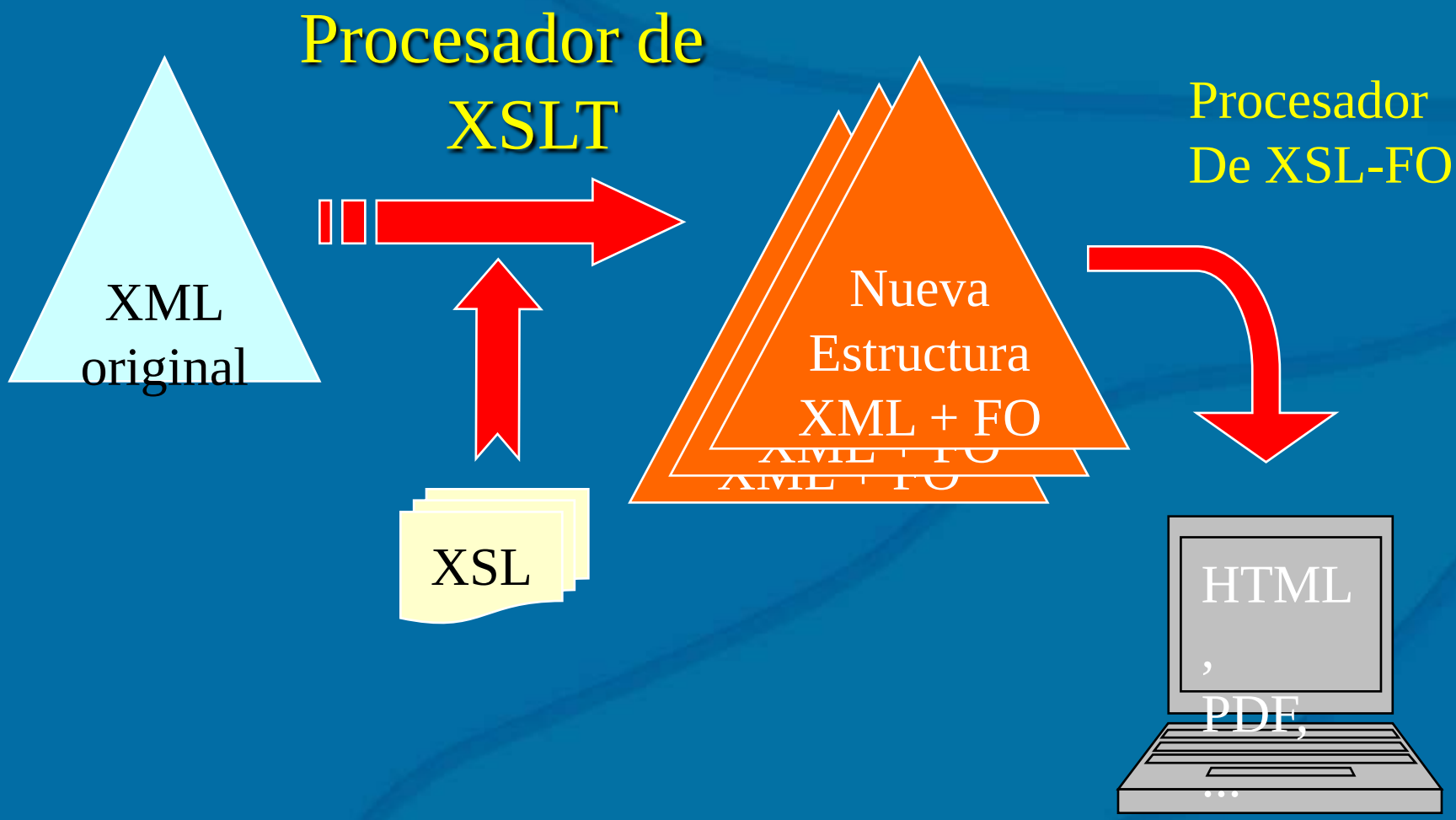
¿ Por que transformar XML ?

- Conversión entre modelos de datos
  - Aunque legible, XML está pensado para el intercambio de información entre aplicaciones.
- Es necesaria una capa de presentación para hacer “amigable” el acceso de los humanos a la información XML.
  - XML produce separación entre el modelo de datos y presentación visual.

¿Cómo funciona?

- Durante el proceso de transformación, XSLT utiliza **XPath** para definir partes del documento fuente que encajan dentro de una o mas plantillas predefinidas
- Por cada coincidencia, XSLT transformará esta parte del documento fuente para generar el documento resultante
- La porción del documento fuente que no encaja con la plantilla permanecerá sin modificación alguna

Proceso de publicación XML-XSL



Ejemplo: Transformando a HTML

poema.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>

<xsl:template match="/">
<html>
 <head>
  <title>Ejemplo</title>
 </head>
 <body>
  <h1>Poema</h1>
 </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Espacio de nombres

Plantilla

Reglas de plantillas

- Las reglas de plantillas (templates) se identifican mediante:

```
<xsl:template match=" . . . ">
...resultado...
</xsl:template>
```

- El valor del atributo match es una expresión XPath
- Cuando el procesador encaja el nodo actual con la expresión genera el valor de la plantilla

Asociar hoja de estilos a documento XML

- Es posible indicar en el documento XML qué hoja de estilos XSL lleva asociada o también la CSS
  - Type: “text/css” (CSS) o “text/xsl” (XSL)”
- Un navegador puede reconocer dicha información y mostrar el resultado de la transformación

```
<?xml-stylesheet type= “text/css” href = “poema.css”?>
<?xml-stylesheet type= “text/xsl” href = “poema.xsl”?>
```

Ejemplo: Asociando una hoja de estilos a un documento XML

Poema.xml

```
<?xml version=”1.0” encoding="ISO-8859-1"?>
<?xml−stylesheet type=”text/xsl” href=”poema.xsl”?>
<poema fecha=”Abril de 1915” lugar=”Granada ”>
<titulo>Alba</titulo>
<verso>Mi corazón oprimido</verso>
<verso>late junto a la alborada</verso>
. . .
</poema>
```

Ejemplo: Obtener valores

poemaLugarFecha.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>

<xsl:template match="poema">
Fecha : <xsl:value-of select="@fecha"/>
,
Lugar : <xsl:value-of select="@lugar"/>
</xsl:template>

</xsl:stylesheet>
```

Ejemplo básico

```
<?xml version= “1.0”?>
  <?xml-stylesheet type= “text/xsl” href =“intro.xsl”?>
     <miCurso>
        <curso>Curso de Lenguajes para Internet</curso>
     </miCurso>
```

```
<?xml version = "1.0"?>
<xsl:stylesheet version ="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match = "miCurso">
    <html>
       <body><p><xsl:value-of select="curso"/></p></body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

```
<html>
<body><p>Curso de Lenguajes para Internet</p></body>
</html>
```

Estructura básica de una hoja XSL

- **Elementos de XSLT**:
  - Pertenecen al *namespace* xsl
  - Sus etiquetas llevan el prefijo xsl: . Son el equivalente a las palabras clave del lenguaje de programación (definidos por el estándar e interpretados por cualquier procesador de XSLT)
- **Elementos LRE** (Literal Result Elements):
  - Son elementos que no pertenecen a XSLT, sino que se repiten en la salida sin más
  - Ejemplo: un elemento <fecha>
- **Elementos de extensión:**
  - Son elementos no-estándar (al igual que los LRE), que son manejados por implementaciones concretas del procesador. Normalmente, no los utilizaremos

<xsl:stylesheet>

- Es el elemento raíz de una hoja XSL
  - version: Suele ser 1.0
  - xmlns:xsl: Asigna el *namespace* xsl (las etiquetas de XSL empiezan por el prefijo xsl:). El valor para XSLT suele ser **http://www.w3.org/1999/XSL/Transform**

```
<xsl:stylesheet version ="1.0"
xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
…
</xsl:stylesheet>
```

Elementos de nivel superior

- Son elementos hijos de xsl:stylesheet
- Son hijos directos (tampoco pueden anidarse)
- Dos excepciones: xsl:variable y xsl:param
- No son instrucciones sobre cómo procesar elementos, sino estructuras contenedoras para instrucciones

xsl:output	xsl:strip-space	xsl:namespace-alias
xsl:template	xsl:preserve-space	xsl:attribute-set
xsl:include	xsl:key	xsl:variable
xsl:import	xsl:decimal-format	xsl:param

<xsl:output>

- Define qué tipo de salida se va a generar como resultado
  - method: xml, html o text
  - encoding: define la forma de representar caracteres que se adoptará en la salida (iso-8859-1, UTF-8, UTF-16, windows-1252)
  - omit-xml-declaration: yes o no. Indica si se genera o no la declaración <?xml...?>
  - indent: yes o no. Si es yes, el procesador (para salidas xml o html) indentará el resultado

Otros elementos de nivel superior

- **<xsl:include>** que permite referenciar plantillas procedentes de una fuente externa
- **<xsl:strip-space>** que elimina antes del procesamiento todos los modos consistentes en espacios en blanco
- **<xsl:preserve-space>** mantiene los espacios en blanco

Reglas de transformación XSLT:Templates

- Es el elemento básico y fundamental del lenguaje de transformación XSLT
- Una **regla** (o "template") consta de dos partes:
  - Una "**etiqueta**" formada por un patrón de localización que selecciona nodos en el árbol XML origen sobre los que se aplica la tranformación
    - Expresión XPath.
  - Una "**acción**" que indica la transformación a realizar sobre los nodos seleccionados.
- Cada hoja de estilo XSL debe contener al menos una regla si se quiere que ejecute algo

Obtención de patrones

- <xsl:template match= “…”>
- <xsl:apply-templates select=“…”>

<xsl:template>

- Cada etiqueta <xsl:template> contiene reglas por aplicar a ciertos elementos de un nodo
  - match es utilizado para asociar la plantilla con un elemento XML. El atributo match también puede utilizarse para aplicarse a una rama completa del documento XML (Por ejemplo match="/" define a todo el documento)
  - name Además de cuando encaja, un template puede invocarse explícitamente (en ese caso se necesita que tenga un nombre)

- Ejemplo: <xsl:template match="/">

Ejemplo

```
<xslTutorial >
<bold>Hello, world.</bold>
<red>I am </red>
<italic>fine.</italic>
</xslTutorial>
```

```
<p> <b>Hello, world.</b></p>
<p style="color:red">I am </p>
<p> <i>fine.</i></p>
```

```
<xsl:stylesheet
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="bold">
     <p><b><xsl:value-of select="."/></b></p>
</xsl:template>
<xsl:template match="red">
    <p style="color:red"><xsl:value-of select="."/></p>
</xsl:template>
<xsl:template match="italic">
    <p><i><xsl:value-of select="."/></i></p>
</xsl:template>
</xsl:stylesheet>
```

Hello, world.
I am
fine.

<xsl:apply-templates>

- Se utiliza para indicar al procesador que intente emparejar templates con cierto nodo o conjunto de nodos (*nodeset*)
  - select: Su valor es una expresión XPath de conjunto de nodos. El procesador intentará emparejar ese conjunto de nodos con sus templates respectivos
- Permite realizar un tratamiento recursivo de todos los elementos del árbol fuente

XPath

- Objetivo: Identificar elementos
- Tienen la forma
  - nodo1/nodo2/.../nodoN
- Ejemplo: persona/apellido/materno
- Describen un camino (*path*)
- Resultado:
  - Un conjunto de nodos
  - String, número o boolean

XPath versus Sistema de archivos

Ficheros y directorios	Nodos dentro de nodos
Respecto a directorio actual	Respecto a nodo actual o de contexto
* = cualquier cosa	* = cualquier nodo
Un fichero por ruta	Uno o varios nodos por ruta

Resumen de sintaxis abreviada XPath

- X – hijo elemento “X”
- * - todos los hijos elemento
- text() - todos los hijos texto
- @Y – atributo “Y”
- X[1] – primer hijo “X”
- X[last()] - último hijo “X”
- */X – nietos “X”
- X//Y – descendientes “Y” de hijo “X”
- //Y – descendientes “Y” de la RAIZ
- //Y/X - descendientes “Y” de HIJO “X”
- .. - padre
- //X[1][@Y=”Z”] - primeros hijos X con atributo Y=”Z”

<xsl:value-of>

- La etiqueta <xsl:value-of> se utiliza para seleccionar el valor de una etiqueta XML y agregarlo al archivo de salida de la transformación
- El valor del atributo **select** contiene una expresión **XPath.** Esta trabaja como la navegación de un sistema de ficheros donde una diagonal vertical delantera (/) selecciona subdirectorios
- **Ejemplo:**

  <xsl:value-of select="catalogo/cd/titulo"/ >

Ordenar nodos: <xsl:sort>

- Permite aplicar un *template* a un conjunto de nodos ordenándolos alfabética o numéricamente
- Sintaxis:

```
<xsl:apply-templates select='XPATH'>
 <xsl:sort select="XPATH"
           data-type="text|number"
           order="ascending|descending"/>
 <xsl:sort … />
</xsl:apply-templates>
```

Ejemplo

```
<source>
<name>Juan</name>
<name>Jaime</name>
<name>Carlos</name>
<name>Alicia</name>
<name>Marta</name>
</source>
```

```
<xsl:stylesheet version = '1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
  <TABLE>
    <xsl:for-each select="//name">
        <xsl:sort order="ascending" select="."/>
          <TR>
            <TH><xsl:value-of select="."/></TH>
          </TR>
        </xsl:for-each>
  </TABLE>
</xsl:template>
</xsl:stylesheet>
```

```
<TABLE>
  <TR>
     <TH>Alicia</TH>
  </TR>
  <TR>
     <TH>Carlos</TH>
  </TR>
  <TR>
     <TH>Jaime</TH>
  </TR>
  <TR>
     <TH>Juan</TH>
  </TR>
  <TR>
     <TH>Marta</TH>
  </TR>
</TABLE>
```

Alicia
Carlos
Jaime
Juan
Marta

Sentencias iterativas: <xsl:for-each>

- La etiqueta XSL **<xsl:for-each>** se utiliza para seleccionar todos los elementos XML del nodo especificado.
- El valor del atributo **select** contiene una expresión **XPath.** Esta trabaja como la navegación de un sistema de ficheros donde una diagonal vertical delantera (/) selecciona subdirectorios.
- **Sintaxis:**

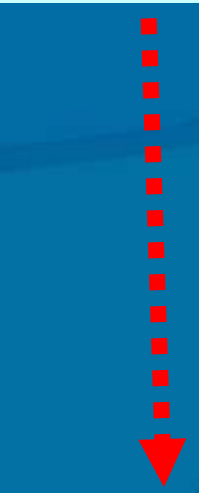
```
<xsl:for-each select="XPATH">
        …
</xsl:for-each>
```

Ejemplo

```
<xslTutorial >
<list> <entry name="A"/>
       <entry name="B"/>
       <entry name="C"/>
       <entry name="D"/>
</list>
</xslTutorial>
```

```
<HTML>
<HEAD> </HEAD>
<BODY> A, B, C, D, </BODY>
</HTML>
```

```
<xsl:stylesheet
xmlns:xsl='http://www.w3.org/1999/XSL/Transform' >
<xsl:template match="list">
       <xsl:for-each select="entry">
              <xsl:value-of select="@name"/>
              <xsl:text> , </xsl:text>
       </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```



A, B, C, D,

Sentencias selectivas: <xsl:if>

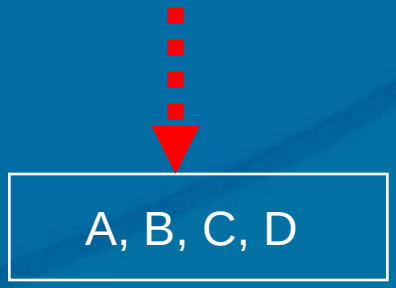
- Permite decidir si se va a procesar o no una parte de la hoja XSL en función de una condición.
- Sintaxis:

```
<xsl:if test='condición'>
…
</xsl:if>
```

- No contiene parte *else*.
- Podemos usar =, !=, >=, > y not( )

Ejemplo

```
<xsl:stylesheet
xmlns:xsl='http://www.w3.org/1999/XSL/Transform' >
<xsl:template match="list">
        <xsl:for-each select="entry">
                <xsl:value-of select="@name"/>
                <xsl:if test="not(position()=last())">
                        <xsl:text> , </xsl:text>
                </xsl:if>
        </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```



Sentencias selectivas: <xsl:choose>

- Permite decidir qué parte de una hoja XSL se debe procesar en función de varias condiciones
- Contiene elementos xsl:when
  - Atributo: test (similar al de xsl:if)
  - Son los diferentes “casos” de una sentencia CASE
- Caso por defecto: xsl:otherwise (sin atributos)
- Sintaxis:

```
<xsl:choose>
        <xsl:when test="condición">
        …
        </xsl:when>
        <xsl:when test="condición">
        …
        </xsl:when>
        <xsl:otherwise>
        …
        </xsl:otherwise>
</xsl:choose>
```

```
<xslTutorial >
<SECTION>
   <DATA>I need a pen.</DATA>
   <DATA>I need some paper.</DATA>
   <SUMMARY>
     I need a pen and some paper
   </SUMMARY>
</SECTION>
<SECTION>
   <DATA>I need bread.</DATA>
   <DATA>I need butter.</DATA>
</SECTION>
</xslTutorial>
```

```
<HTML> <HEAD> </HEAD>
<BODY>
<P>SUMMARY: I need a pen
and some paper</P>
<P>DATA: I need bread.</P>
<P>DATA: I need butter.</P>
</BODY> </HTML>
```

SUMMARY: I need a pen
and some paper
DATA: I need bread.
DATA: I need butter.

```
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' >
 <xsl:template match="//SECTION">
  <xsl:choose>
    <xsl:when test='SUMMARY'>
    <P><xsl:text>SUMMARY:</xsl:text><xsl:value-of select="SUMMARY"/></P>
    </xsl:when>
    <xsl:otherwise>
      <xsl:for-each select="DATA">
         <P><xsl:text> DATA: </xsl:text> <xsl:value-of select="."/></P>
      </xsl:for-each>
    </xsl:otherwise>
  </xsl:choose>
 </xsl:template>
</xsl:stylesheet>
```

Construcción de elementos en el árbol resultado

- ¿Cómo generar un elemento con cierta etiqueta y “construir” sus atributos?
- A veces la sintaxis no nos lo permite directamente

  Ej. **<BODY BGCOLOR=“<xsl:value-of select=“color-elegido”/>”>**
- Se pueden utilizar los llamados **AVT (Attribute Value Template):** las expresiones entre llaves se evalúan como si hubiera un value-of
  - Para poner llaves "de verdad", poner cada una dos veces
- Se pueden necesitar instrucciones para “construir” dichos elementos
- **xsl:element**: Construye un elemento en el árbol resultado
  - Atributos: name
- **xsl:attribute**: añade un atributo al elemento
  - Atributos: name
  - El valor está encerrado como texto libre dentro de xsl:attribute

Ejemplo

```
<BODY BGCOLOR="#00FFFF">
<P>Esto es una prueba</P>
</BODY>
```

```
<xsl:element name="BODY">
        <xsl:attribute name="BGCOLOR">
                #00FFFF
        </xsl:attribute>
        <xsl:element name="P">
                Esto es una prueba
        </xsl:element>
</xsl:element>
```

Definición de variables

- Permiten asignar valores a etiquetas para usarlos cómodamente
- Sintaxis:

```
<xsl:variable name="nombre" select="XPATH">
```

- No existe el concepto de variable global. La acumulación de valores debe hacerse mediante llamadas recursivas a funciones.
- Se definen dentro de <*xsl:template*> y NO conservan el valor de una llamada a otra del *template*.

Llamada a funciones

- La etiqueta <xsl:param> permite definir parámetros con los que posteriormente se llamará a la función.
- La llamada a la función se hará según:

```
<xsl:call-template name="nombre-función">
 <xsl:with-param name="par1" select="valor"/>
 <xsl:with-param name="par2" select="valor"/>
</xsl:call-template>
```

- Como ya se ha indicado, se permite hacer llamadas recursivas a funciones

auction1.xml - Notepad

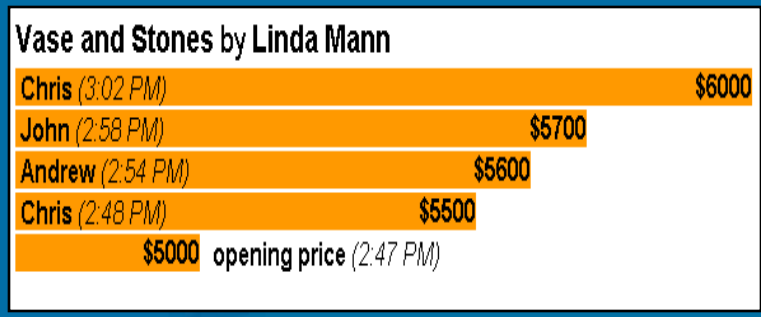
File Edit Format Help

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

  <ITEM>
    <TITLE>Vase and Stones</TITLE>
    <ARTIST>Linda Mann</ARTIST>
    <DIMENSIONS>20x30 inches</DIMENSIONS>
    <MATERIALS>Oil</MATERIALS>
    <YEAR>1996</YEAR>
    <DESCRIPTION>Still Life</DESCRIPTION>
    <PREVIEW-SMALL src="burl-s.jpg" width="300"
height="194" alt="Vase and Stones"/>
    <BIDS>
      <BID>
        <PRICE>6000</PRICE>
        <TIME>3:02:22 PM</TIME>
        <BIDDER>Chris</BIDDER>
        <TIMESTAMP>1307</TIMESTAMP>
      </BID>
      <BID>
        <PRICE>5700</PRICE>
        <TIME>2:58:42 PM</TIME>
        <BIDDER>John</BIDDER>
        <TIMESTAMP>1315</TIMESTAMP>
      </BID>
      <BID>
        <PRICE>5600</PRICE>
        <TIME>2:54:32 PM</TIME>
        <BIDDER>Andrew</BIDDER>
        <TIMESTAMP>1308</TIMESTAMP>
      </BID>
      <BID>
        <PRICE>5500</PRICE>
        <TIME>2:48:08 PM</TIME>
        <BIDDER>Chris</BIDDER>
        <TIMESTAMP>1307</TIMESTAMP>
      </BID>
      <BID>
        <PRICE>5000</PRICE>
        <TIME>2:47:58 PM</TIME>
        <BIDDER>opening price</BIDDER>
        <TIMESTAMP>1298</TIMESTAMP>
```

tabla.xsl

Price	Time	Bidder
$6000	3:02:22 PM	Chris
$5700	2:58:42 PM	John
$5600	2:54:32 PM	Andrew
$5500	2:48:08 PM	Chris
$5000	2:47:58 PM	opening price

barra.xsl



arte.xsl

auction1.xml - Notepad

File Edit Format Help

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

  <ITEM>
    <TITLE>Vase and Stones</TITLE>
    <ARTIST>Linda Mann</ARTIST>
    <DIMENSIONS>20x30 inches</DIMENSIONS>
    <MATERIALS>Oil</MATERIALS>
    <YEAR>1996</YEAR>
    <DESCRIPTION>Still Life</DESCRIPTION>
    <PREVIEW-SMALL src="burl-s.jpg" width="300"
height="194" alt="Vase and Stones"/>
    <BIDS>
      <BID>
        <PRICE>6000</PRICE>
        <TIME>3:02:22 PM</TIME>
        <BIDDER>Chris</BIDDER>
        <TIMESTAMP>1307</TIMESTAMP>
      </BID>
      <BID>
        <PRICE>5700</PRICE>
        <TIME>2:58:42 PM</TIME>
        <BIDDER>John</BIDDER>
        <TIMESTAMP>1315</TIMESTAMP>
      </BID>
      <BID>
        <PRICE>5600</PRICE>
        <TIME>2:54:32 PM</TIME>
        <BIDDER>Andrew</BIDDER>
        <TIMESTAMP>1308</TIMESTAMP>
      </BID>
      <BID>
        <PRICE>5500</PRICE>
        <TIME>2:48:08 PM</TIME>
        <BIDDER>Chris</BIDDER>
        <TIMESTAMP>1307</TIMESTAMP>
      </BID>
      <BID>
        <PRICE>5000</PRICE>
        <TIME>2:47:58 PM</TIME>
        <BIDDER>opening price</BIDDER>
        <TIMESTAMP>1298</TIMESTAMP>
```

ie5.xsl
ie4.xsl
nav3.xsl

nokia.xsl
sony.xsl

edi_x.xsl
sap_y.xsl
flat_z.xsl

Lenguajes para Internet

Juan Manuel Cueva Lovelle **OOTLab**

Procesadores XSLT

- Aplicación de una hoja de estilo a una fuente XML para producir una salida
  - ***MSXML3 SP4 (Internet Explorer)***
    http://www.microsoft.com/xml
  - ***Saxon***
    http://users.iclway.co.uk/mhkay/saxon/
  - ***Xalan***
    http://xml.apache.org/xalan/overview.html

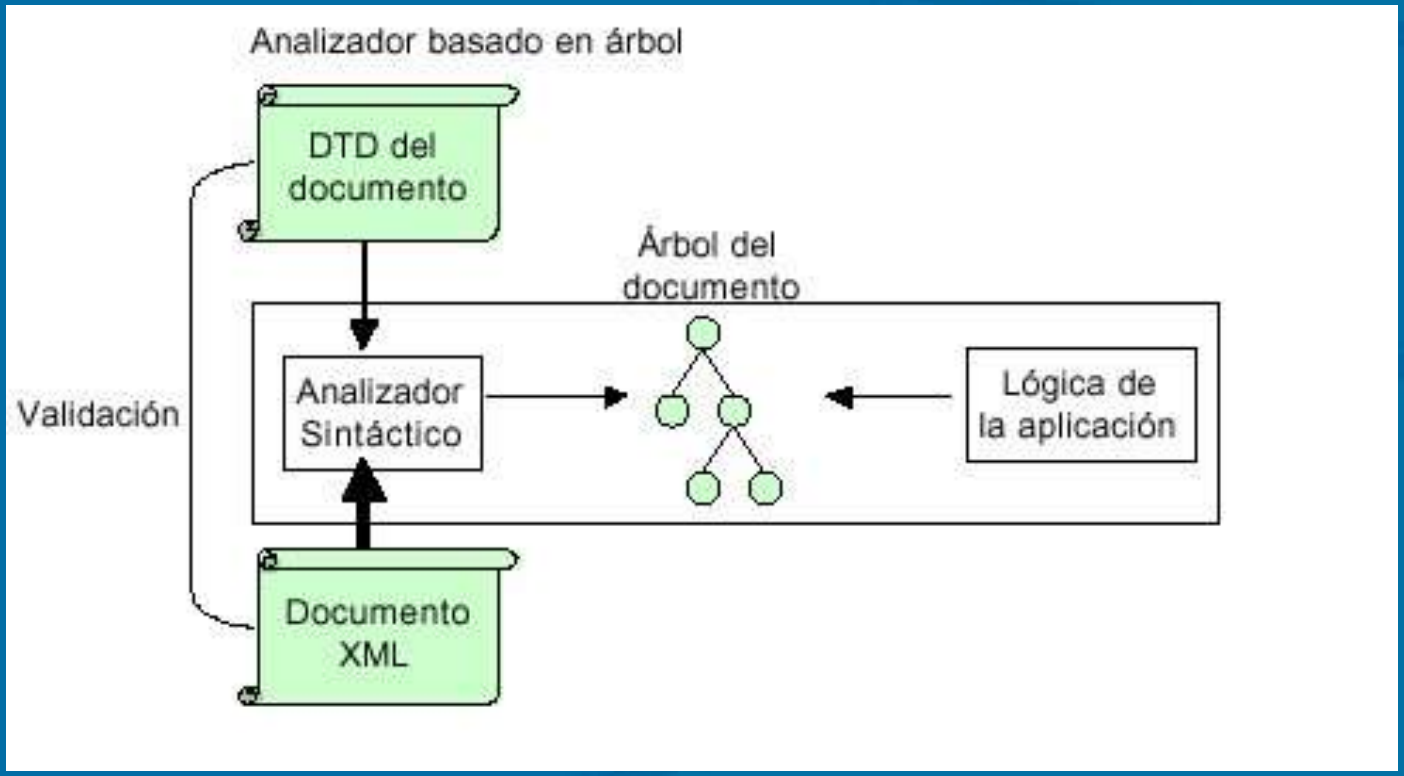
Referencias

- W3Schools: http://www.w3schools.com
- El sitio de XSLT: http://www.xslt.com
- Especificación XSL. http://www.w3.org/Style/XSL
- Especificación XSLT. http://www.w3.org/TR/xslt

Parsers XML

Procesamiento de XML

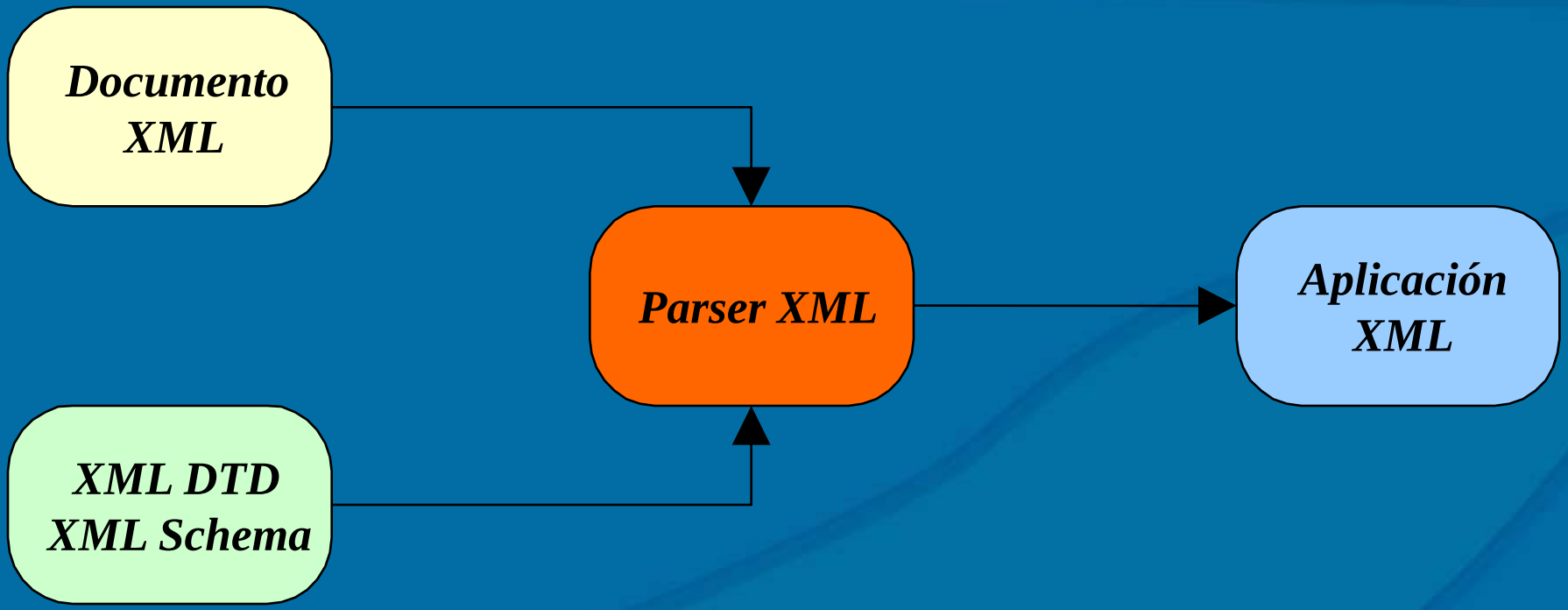
Procesamiento de XML



¿Qué es un parser XML?

- Un **parser** o **procesador** o **analizador sintáctico** procesa el documento XML y verifica que es XML bien formado (y/o válido)
- Es la herramienta principal de cualquier aplicación XML. Podemos incorporarlos a nuestras aplicaciones, de manera que estas puedan manipular y trabajar con documentos XML

Parsing



Tipos de parsers

- **Sin validación**: chequea que el documento esté **bien formado** de acuerdo a las reglas de **sintaxis** de XML
- **Con validación**: además de comprobar que el documento está bien formado, comprueba que éste sea válido utilizando un **DTD**

En el caso de utilizar un DTD, es preferible utilizar un parser con validación

Parsers

- Los navegadores incluyen sus propios parsers.
  - Microsoft Internet Explorer 5 (o superior), que utiliza el parser de MS (incluido en la librería MSXML.DLL)
  - Mozilla, que internamente utiliza el parser EXPAT (escrito en C)
- DOM y SAX son parsers XML que comprueban que el documento esté bien formado y válido

SAX y DOM

- Las APIs de SAX y DOM están estandarizadas y existen un gran número de implementaciones para distintos lenguajes (C++, **Java**, C#,etc.)
  - Ej.: Apache Software Foundation proporciona **Crimson** (SAX y DOM sólo para Java), **Xerces** (SAX y DOM) y **Xalan** (XSL)
  - En el caso de Java, familia de paquete **org.xml.sax** y **org.w3c.dom** (básicamente contienen interfaces y clases abstractas)
  - Lo que no está estandarizado es cómo crear instancias de los parsers

JAXP (Java API for XML Processing)

- Forma parte de J2SE 1.4
- Define un API para trabajar con parsers SAX, DOM y transformaciones XSL
  - Proporciona factorías para crear instancias de parses y transformadores XSL de manera portable
- Sun proporciona una implementación de JAXP para versiones anteriores a J2SE 1.4
  - Incluye las APIs **org.xml.sax**, **org.w3c.dom** y **javax.xml.{parsers, transform}**
  - Incluye Crimson y Xalan como implementaciones por defecto
  - Se pueden usar otras implementaciones vía configuración

Lenguajes derivados de XML

- MathML
  - Visualización de ecuaciones matemáticas
- SVG
  - Gráficos vectoriales
- SMIL
  - Presentaciones multimedia
- P3P
  - Descripción de características de privacidad
- WML
  - Similar a HTML para teléfonos móviles
- VoiceML
  - Portales basados en voz
- XML Signature
  - • Firma de recursos Web
- XKMS
  - Firmas y criptografía
- XML Query
  - Consultas de documentos (Bases de datos)
- XBRL
  - Contabilidad
- • ebXML
  - Negocios electrónicos (e-business)
- • SyncXML
  - Sincronización de dispositivos
- • UPnP
  - Plug and Play universal

Tecnología XML. Referencias

- ***Libro recomendado***
  - ***XML imprescindible***. E.R. Harold y W.S. Means. Anaya-O'Reilly, 2005.
- http://www.w3.org
  - Página web donde están los estándares de Internet
  - Oficina española
    - www.w3c.es
- http://www.xml.org
  - El portal de XML para la industria
- http://topxml.com
  - Artículos y software XML

Referencias Generales

- http://www.di.uniovi.es/~cueva/xml/CursoXML.html
  - Página Web del autor con ejemplos y recursos XML
- Guías breves de tecnologías W3C.
  - Oficina Española W3C
  - www.w3c.es
- Programación de Aplicaciones Web
  - S. Rodríguez de la Fuente et al.
  - Editorial Thomson, Madrid, 2003