

Lenguaje XML

Jose Emilio Labra Gayo
Departamento de Informática
Universidad de Oviedo

¿HTML como formato de representación?

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

<h1>Pizzería Al Capone</h1>
<table>
<caption>Lista de Pizzas</caption>
<tr>
<td>Barbacoa</td>
<td>Mozzarella, Queso, Bacon</td>
<td>7&euro;</td>
</tr>
...
</body >
</html >
```

En HTML, las marcas tienen un significado predefinido

Mezcla información de la pizza con presentación en tabla

```
<pizza nombre="Barbacoa">
<ingredientes nombres="Mozzarella Queso Bacon" />
<precio moneda="euro" valor="7" />
</pizza>
```

Posteriormente, podría representarse en una tabla...

O en otros formatos no previstos inicialmente: Estadísticas, WAP, TV, ...

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
 - HTML es un subconjunto de SGML
 - Problema de SGML: Demasiado complicado para su adopción en la Web
- XML
 - Desarrollado por el consorcio Web (1995)
 - Versión simplificada de SGML
 - 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

Ejemplo de XML

Las marcas tienen un significado propio de la aplicación

pizzas.xml

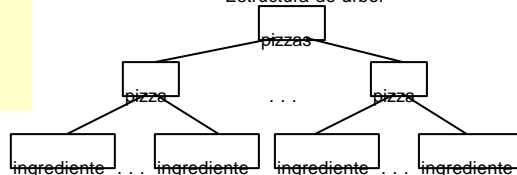
```
<?xml version="1.0"?>
<!DOCTYPE pizzas SYSTEM "pizzas.dtd">
<pizzas>
  <pizza nombre="Barbacoa" precio="8">
    <ingrediente nombre="Salsa Barbacoa" />
    <ingrediente nombre="Mozzarella" />
    <ingrediente nombre="Pollo" />
    <ingrediente nombre="Bacon" />
    <ingrediente nombre="Tenera" />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

DTD = Declaración de Tipo de Documento

pizzas.dtd

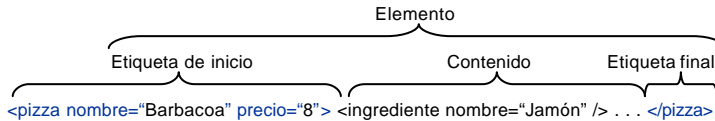
```
<!ELEMENT pizzas (pizza*)>
<!ELEMENT pizza (ingrediente*)>
<!ELEMENT ingrediente (#PCDATA)>
<!ATTLIST pizza nombre CDATA #REQUIRED>
<!ATTLIST pizza precio CDATA #REQUIRED>
<!ATTLIST ingrediente nombre CDATA #REQUIRED>
```

Estructura de árbol



Definición de XML

- XML se basa en la utilización de elementos
- Un elemento está formado por:
 - Una **etiqueta inicial** (nombre entre signos < y >): `<etiqueta>`
 - La etiqueta inicial puede contener **atributos** : `<etiqueta atributo="valor">`
 - 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" />
```

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

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

Otras características de XML

- Comentarios
 - `<!-- el texto de un comentario no es analizado -->`
- Secciones CDATA
 - Si se desea introducir código sin analizar

```
<codigo_HTML>
<html>
<body >Hola</body >
</html>
</codigo_HTML>
```

```
<codigo_HTML>
<![CDATA[
<html>
<body >Hola</body >
</html>
]]>
</codigo_HTML>
```

Declaración de tipo de documento (DTD)

DTD interno

```
<?xml version="1.0"?>
<!DOCTYPE pizzas [
  <ELEMENT pizzas (pizza*)>
  . . .
]>
<pizzas> . . . </pizzas>
```

DTD externo

SYSTEM (DTDs de ámbito local)

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

PUBLIC (DTDs compartidos por diversas organizaciones)

```
<?xml version="1.0"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.0//EN"
  "http://www.w3c.org/TR/REC-html/strict.dtd">
```

DTD: Tipos de declaraciones

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

(?)	= 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)>
```

- 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" />
```

- Entidades: Asignan nombres a ciertos elementos (similar a variables)
 - Se denotan por &entidad;

```
<!ENTITY mezcla "Mezcla de 4 quesos">
```

```
<pizza nombre="4 Quesos" precio="7">
<ingrediente nombre="&mezcla;" />
</pizza>
```

- Entidades predefinidas: Permiten incluir etiquetas sin analizar

```
&lt; < &quot; " &apos; '
&gt; > &amp; &
```

- Permiten usar archivos externos (Incluir otros documentos XML)

pizzas.xml

```
<pizzas>
<pizza nombre="4 Quesos" precio="7">
<ingrediente nombre="Jamón" />
<ingrediente nombre="Queso" />
</pizza>
...
</pizzas>
```

personal.xml

```
<personal>
<trabajador
nombre=" Benito Alcaparra" >
...
</trabajador>
...
</personal>
```

establecimiento.dtd

```
<!ELEMENT establecimiento ANY>
<!ENTITY personal SYSTEM "personal.xml">
<!ENTITY pizzas SYSTEM "pizzas.xml">
```

establecimiento.xml

```
<establecimiento
nombre="Pizzería Al Capone">
&personal;
&pizzas;
</establecimiento>
```

- También se pueden incluir archivos externos de formatos binarios

```
<!NOTATION gif SYSTEM "gifEditor.exe">
<!ENTITY dibujo SYSTEM "logotipo.gif" NDATA gif>
```

- Entidades parámetro: Permiten dar nombres a partes de un DTD
 - Se denotan por %entidad;

```
<!ENTITY establecimiento (nombre,dueño?,calle,número?,ciudad,país,códigoPostal)>
<!ENTITY persona (dni, nombre,calle,número?,ciudad,país,códigoPostal)>
```

```
<!ENTITY %localización "calle,número?,ciudad,país,códigoPostal" >
<!ENTITY establecimiento (nombre,dueño?,%localización;)>
<!ENTITY persona (dni, nombre, %localización;)>
```

- Entidades externas: Permiten incluir elementos externos en una DTD
 - Aplicación: Dividir la definición de una DTD en varios documentos

```
<!ENTITY %persona SYSTEM "persona.dtd">
<!ENTITY %establecimiento SYSTEM "establecimiento.dtd">

%persona;
%establecimiento;
```



- Creación de ficheros XML y validación
- Procesadores de XML
 - Chequean que está bien formado
 - Validan
- Productos
 - Visuales: XML Writer, XML Spy, ...
 - Modo texto: xmllint, msxml, ...
- xmllint forma parte de la librería libxml de GNOME

```
xmllint --valid --noout fichero.xml
```

Validar
Si no se pone nada,
Chequea que está bien formado

No muestra resultado
Si no hay mensajes ⇒ OK

XML: Diseño de vocabularios

- Separación tradicional de dos mundos
 - Sistemas orientados a Datos
 - Información uniforme y fuertemente estructurada (ej. Tablas)
 - Mucha cantidad de información repetida
 - Objetivo: Procesamiento eficiente (Almacenes de datos)
 - Sistemas orientados a Documentación
 - Información poco uniforme y entrelazada (ej. Libros)
 - No existe un patrón uniforme
 - Objetivo: Comunicación, Presentación (Mensajes)
- Se podría añadir un tercer mundo:
 - Programación Orientada a Objetos
 - Propuestas para añadir capacidad de programación a documentos
- XML: Información semi-estructurada (Lugar intermedio)
 - Estructuras jerárquicas entrelazadas

XML: Diseño de vocabularios

- Características a tener en cuenta
 - Tamaño de documentos
 - Facilidad de escritura
 - Facilidad de procesamiento
 - Flexibilidad (ej. HTML es muy flexible, Bases de Datos = menos)
 - Consistencia: Evitar características incoherentes
 - Nivel de abstracción: Buscar término medio en nivel de detalle
 - `<fecha>10 Marzo 2003</fecha>`
 - `<fecha><día>10</día><mes>Marzo</mes><año>2003</año></fecha>`
- Patrones de diseño:
 - <http://www.xmlpatterns.com>

XML: Diseño de vocabularios

Representación de propiedades

```
<pizza
  nombre="Margarita"
  precio="6" />
```

¿Atributos o Elementos?

```
<pizza>
  <nombre>Margarita </nombre>
  <precio>6</precio>
</pizza>
```

Razones filosóficas:

Atributos: valores asociados con objetos sin identidad propia (edad)
Subelementos: valores con identidad propia (fecha-nacimiento)



Orígenes (SGML):

Atributos: meta-información (información sobre el contenido)
Subelementos: Contenido



XML: Diseño de vocabularios

Representación de propiedades

```
<pizza
  nombre="Margarita"
  precio="6" />
```

¿Atributos o Elementos?

```
<pizza>
  <nombre>Margarita </nombre>
  <precio>6</precio>
</pizza>
```

- Pueden incluirse restricciones sobre su valor
 - Ej. valor "si" o "no"
- Pueden definirse valores por defecto
- Pueden validarse los valores ID e IDREF
- Ocupan menos espacio
- Pueden definirse restricciones sobre espacios en blanco (NMTOKENS)
- Más fáciles de procesar (SAX y DOM)
- Acceso a entidades externas (datos binarios)

- Soportan valores arbitrariamente complejos y repetidos
- Establecen un orden
- Soportan *atributos de atributos*
- Mayor flexibilidad ante modificaciones



Espacios de Nombres

- XML NameSpaces permite especificar espacios de nombres para evitar colisiones de identificadores

```
<establecimiento>
<nombre> Pizzería Al Capone </nombre>
<direccion>C/ Génova Nº 3,Oviedo, España</direccion>
<teléfono>985203040 </teléfono>
</establecimiento>
```

```
<persona>
<nombre> Vito Corleone</nombre>
<teléfono>985223344 </teléfono>
<dni> 98765432 </dni>
</persona>
```

...y si queremos añadir información del dueño?

```
<establecimiento>
<nombre> Pizzería Al Capone </nombre>
<direccion>C/ Génova Nº 3,Oviedo, España </direccion>
<teléfono>985203040 </teléfono>
<dueño>
  <persona>
    <nombre> Vito Corleone</nombre>
    <teléfono>985223344 </teléfono>
    <dni> 98765432 </dni>
  </persona>
</dueño>
</establecimiento>
```

<nombre> se refiere a persona o a establecimiento?

Espacios de Nombres

- Un alias se crea asignando un nombre a una URL
 - El ámbito del alias abarca al nodo y a sus hijos

```
<alias:etiqueta xmlns:alias="direccion URL">
  <alias.subetiquetas />
</alias:etiqueta>
```

- Espacio por defecto (no se pone alias)

```
<etiqueta xmlns="direccion URL">
  <subetiquetas />
</etiqueta>
```

Permite
diferenciar
nombres de
persona y
establecimiento

```
<e:establecimiento xmlns:e="establecimiento.dtd"
  xmlns:p="persona.dtd">
  <e:nombre>Pizzería Al Capone</e:nombre>
  <e:direccion>C/ Génova Nº 3,Oviedo, España</e:direccion>
  <e:telefono>985203040</e:telefono>
  <e:dueño>
    <p:persona>
      <p:nombre>Vito Corleone</p:nombre>
      <p:telefono>985223344</p:telefono>
      <p:dni> 98765432</p:dni>
    </p:persona>
  </e:dueño>
</e:establecimiento>
```

```
<e:establecimiento xmlns:e="establecimiento.dtd">
  <e:nombre>Pizzería Al Capone</e:nombre>
  <e:direccion>C/ Génova Nº 3,Oviedo, España</e:direccion>
  <e:telefono>985203040</e:telefono>
  <e:dueño>
    <p:persona xmlns:p="persona.dtd">
      <p:nombre>Vito Corleone</p:nombre>
      <p:telefono>985223344</p:telefono>
      <p:dni> 98765432</p:dni>
    </p:persona>
  </e:dueño>
</e:establecimiento>
```

```
<e:establecimiento xmlns:e="establecimiento.dtd">
<e:nombre>Pizzería Al Capone</e:nombre>
<e:direccion>C/ Génova Nº 3,Oviedo, España</e:direccion>
<e:teléfono>985203040</e:teléfono>
<e:dueño>
  <persona xmlns ="persona.dtd">
    <nombre>Vito Corleone</nombre>
    <teléfono>985223344</teléfono>
    <dni> 98765432</dni>
  </persona>
</e:dueño>
</e:establecimiento>
```

- 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 chequeo de restricciones numéricas

```

<?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>

```

Permite especificar
rangos de inclusión

Permite especificar
tipos

Asociación del fichero
XML con el esquema

pizzas.xml

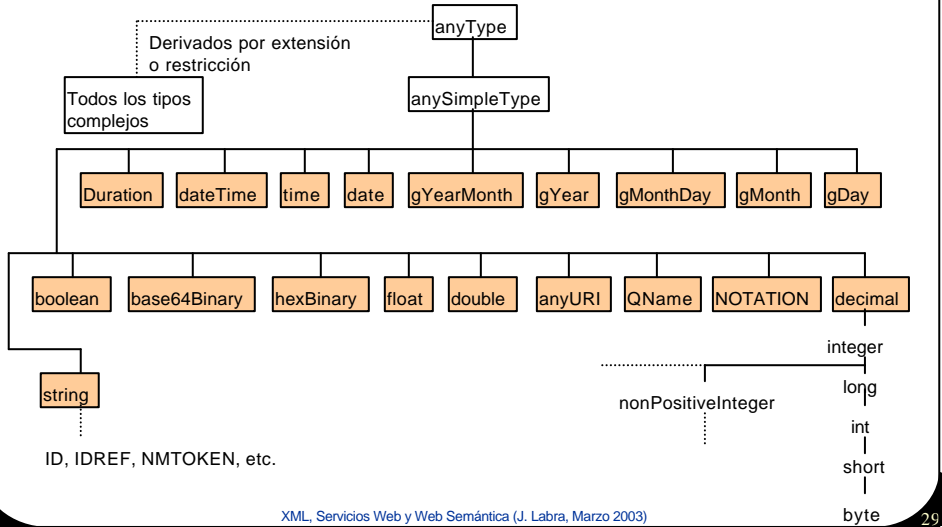
```

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

```

- Tipos de datos Primitivos
 - Se define un sistema completo de tipos de datos
- Tipos de datos primitivos
 - Diferencia entre espacio de valores y espacio léxico
 - 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 (contable, no contable)
 - *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

- Jerarquía de tipos



- El usuario puede definir nuevos tipos simples
- Ejemplos:

Enumeración

```
<xs:simpleType name="género">
  <xs:restriction base="xs:string">
    <xs:enumeration value="hombre"/>
    <xs:enumeration value="mujer"/>
  </xs:restriction>
</xs:simpleType>
```

Restricción de longitud

```
<xs:simpleType name="colorRGB">
  <xs:restriction base="xs:unsignedByte">
    <xs:length value="3"/>
  </xs:restriction>
</xs:simpleType>
```

Unión de dos tipos

```
<xs:simpleType name="color">
  <xs:union memberTypes="colorRGB">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="rojo"/>
        <xs:enumeration value="azul"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

- Permiten combinar varios elementos y atributos en un tipo de datos
- Contenido:
 - *empty*: Elemento vacío
 - *simpleContent*: Contenido sin subelementos
 - *complexContent*: Puede incluir subelementos conectados mediante:
 - *sequence*: secuencia ordenada
 - *choice*: Elementos alternativos
 - *all*: Secuencia no ordenada
 - *group*: permite dar nombre a un grupo o hacer referencia a otro grupo (reutilización)
 - Puede especificarse contenido mixto mediante: *mixed*="True"

- Otras características
 - Soporte para espacios de nombre múltiples
 - Valores nulos, unicidad y claves
 - Mecanismos de reutilización
 - Grupos de atributos
 - Grupos de sustitución: Especie de *alias* de elementos
 - Tipos abstractos: sólo se usan para derivar otros tipos
 - Inclusión e importación de otros esquemas
 - Redefinición (similar a inclusión pero permite modificar)
 - Subtipos de instancias
- Restricciones:
 - No soporta entidades (se necesita DTD)
 - Lenguaje de Restricciones limitado
 - Existen propuestas como: Schematron, Relax-NG, etc.
 - No se pueden definir elementos sensibles al contexto
 - Tamaño de archivos XML Schema puede ser excesivo

- Define cómo acceder a nodos de un documento XML
- Forma parte importante de XSLT y Xquery
- Ejemplo: Acceder al atributo ISBN del elemento libro:
`libro/autor/@nombre`
- Puede devolver conjuntos de nodos (node sets)
Ejemplo: Devolver apellidos de autores de un libro
`libro/autores/autor/apellido`
- Pueden definirse filtros dentro de una especificación
`libro/autores/autor[1]/apellido`

Expresiones de XPath

<code>a/b</code>	todos los <code></code> hijos directos de <code><a></code>
<code>a//b</code>	todos los <code></code> descendientes de <code><a></code>
<code>/</code>	nodo raíz
<code>.</code>	nodo actual
<code>..</code>	nodo padre del nodo actual
<code>*</code>	cualquier nodo
<code>a b</code>	elementos <code><a></code> o <code></code>
<code>a/b[1]</code>	primer <code></code> hijo directo de <code><a></code>
<code>//b</code>	todos los <code></code> en cualquier parte del documento
<code>./b</code>	todos los <code></code> descendientes del nodo actual
<code>a[@b]</code>	todos los <code><a></code> con el atributo <code>b</code>
<code>a[@b="hola"]</code>	todos los <code><a></code> cuyo atributo <code>b</code> valga "hola"
<code>\$a</code>	valor de la variable <code>a</code>

XPath incluye funciones para manipular datos de tipo booleano, numérico y cadenas de caracteres

Enlaces y referencias en XML

- Enlaces de HTML

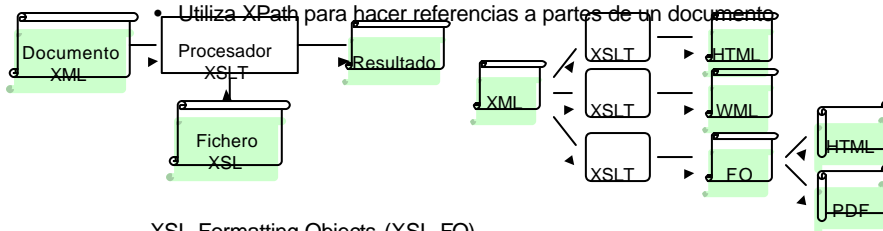
```
<a href="http://www.uniovi.es">Universidad de Oviedo</a>
```

- Limitaciones
 - Empotrados en código fuente
 - Sólo permiten navegación en una dirección
 - Sólo conectan 2 recursos
 - No especifican el comportamiento del navegador
- XLink + XPointer = Solución aportada por XML

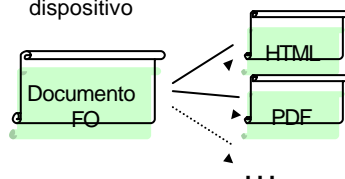
```
<patrocinadores xlink:href="patrocinadores.html"
  xlink:role="mostrar lista de patrocinadores"
  xlink:title="Lista de patrocinadores"
  xlink:show="new"
  xlink:actuate="onRequest"/>
```

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 formato independientes del dispositivo



- El documento XML se puede asociar a una transformación XSLT
- Algunos visualizadores, al recibir un documento XML, transforman el documento y visualizan el resultado

```
<?xml version="1.0"?>
<!DOCTYPE pizzas SYSTEM "pizzas.dtd">
<?xml-stylesheet type="text/xsl" href="pizzas.xsl" ?>
<pizzas>
...
</pizzas>
```

- XSLT es un lenguaje declarativo (transforma un árbol en otro árbol)
- El programador incluye una serie de reglas de transformación
- El procesador es el que se encarga de obtener el árbol y de escribir el resultado
- Las reglas se basan en la definición de plantillas (*templates*)
 - Las plantillas utilizan sintaxis de XPath

```
<xsl:template match="valor a encajar">
    código de salida
</xsl:template>
```

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

```
<xsl:template match="/">
<html><body ><h1>Pizzas del Restaurante Al Capone</h1>
<xsl:apply-templates />
</body ></html>
</xsl:template>
```

Valores que se incluyen en resultado

Patrón de encaje

```
<xsl:template match="pizzas">
<table><caption>Tipos de Pizzas</caption><tr>
<xsl:apply-templates /></td>
</table>
</xsl:template>
```

Referencia a valor de atributo

```
<xsl:template match="pizza">
<tr><td><xsl:value-of select="@nombre"/></td>
<td><xsl:apply-templates /></td>
<td><xsl:value-of select="@precio" /></td></tr>
</xsl:template>
```

```
<xsl:template match="ingrediente"><xsl:value-of select="@nombre" />
</xsl:template>
</xsl:stylesheet>
```

- El procesador XSLT recorre el árbol desde la raíz
 - Recorre los elementos padre antes que los hijos
 - Para cada elemento, si existe una plantilla aplicable, se aplica y ya no se examinan más elementos descendientes (salvo que se solicite)
 - Principales Instrucciones
 - `<xsl:apply-templates select="Patrón de XPath">`
Solicita que continúe aplicando plantillas
 - `<xsl:value-of select="Expresión de XPath" />`
Generar un valor a partir de la expresión
 - `<xsl:for-each select="Expresión de XPath" />`
Para iterar sobre la serie de valores de la expresión
 - `<xsl:copy-of select="Expresión de XPath" />`
Copiar nodos del árbol

- Instrucciones condicionales

<code><xsl:if></code>	Condición simple
<code><xsl:choose></code>	Condición múltiple (case)
<code><xsl:when></code>	Elementos del condicional múltiple
<code><xsl:otherwise></code>	Valor por defecto

```
<xsl:choose>
<xsl:when test="$contador = 2">...
<xsl:when test="$contador = 2">...
<xsl:otherwise>...
</xsl:choose>
```

- Generación de nodos XML

<code><xsl:number></code>	Añade un número
<code><xsl:attribute></code>	Añade un atributo
<code><xsl:element></code>	Añade un elemento
<code><xsl:comment></code>	Añade un comentario
<code><xsl:processing-instruction></code>	Genera una instrucción de procesamiento
<code><xsl:text></code>	Genera un nodo de texto

```
<xsl:element name="a">
<xsl:attribute name="href">
#<xsl:value-of select="@id">
<xsl:value-of select="@nombre">
</xsl:attribute>
</xsl:element>
```

```
<persona id="id1" nombre="pepe" />
```

```
<a href="#id1">pepe</a>
```

- Otras instrucciones

`<xsl:variable>` Declara una variable

XSLT es un lenguaje declarativo
No hay asignación destructiva! ☺

```
<xsl:variable name="tamaño">5
</xsl:variable>

<xsl:if test="$tamaño = '5'"> . . .
</xsl:if >
```

`<xsl:sort>`

Clasificar nodos

`<xsl:include>`

Incluir otra hoja de estilos

`<xsl:import>`

Importar otra hoja

`<xsl:call-template>`

Llamar a otra plantilla

`<xsl:param>`

Declara el valor por defecto de un parámetro

`<xsl:with-param>`

Asignar un valor a un parámetro

Enumerar las pizzas con un índice hipertextual, ordenarlas por precio y mostrar la media de los precios,

The screenshot shows a web browser window displaying a page titled "Pizzas del Restaurante Al Capone". The page content includes a list of pizza types: "Tipos de pizza: 1. Barbacoa | 2. Hawaiana | 3. 4 Quesos | 4. Margarita". Below this is a table with columns "Pizza", "Ingredientes", and "Precio". The table lists four pizzas: Margarita (price 6), Hawaiana (price 7), 4 Quesos (price 7), and Barbacoa (price 8). At the bottom, it shows "Media de precios = 7".

Callouts with arrows pointing to specific elements on the page:

- "Enumerar las pizzas" points to the list of pizza types.
- "Incluir un índice al principio" points to the list of pizza types.
- "Ordenar por precio" points to the table of pizzas.
- "Calcular la media de los precios" points to the "Media de precios = 7" text.

```

<xsl:template match="/">
...
<body ><h1>Pizzas del Restaurante Al Capone</h1>
  <xsl:apply-templates mode="cabecera" />
  <xsl:apply-templates />
</body ></html>
</xsl:template>

<xsl:template match="pizzas" mode="cabecera">
  <div class="header">Tipos de pizza:
  <xsl:for-each select="pizza">
  <xsl:number value="position()" format="1. " />
  <xsl:element name="a">
    <xsl:attribute name="href" >#P<xsl:number level="single"/></xsl:attribute>
    <xsl:value-of select="@nombre"/></xsl:element > |
  </xsl:for-each>
  </div><hr/>
</xsl:template>
...

```

Dos modos de recorrido

Numera las pizzas

Genera una lista de enlaces de la forma
Margarita

```

<xsl:template match="pizzas">
  <table border="1"><caption>Tipos de Pizzas</caption>
  <tr><th>Pizza</th><th>Ingredientes</th><th>Precio</th></tr>
  <xsl:for-each select="//pizza">
    <xsl:sort data-type="number" select="@precio" />
    <xsl:apply-templates select="."/>
  </xsl:for-each>
</table>
<p>Media de precios = <xsl:value-of select="sum(//@precio) div count(//@precio)" /></p>
</xsl:template>

<xsl:template match="pizza">
  <tr><td><xsl:element name="a">
    <xsl:attribute name="name">P<xsl:number level="single"/></xsl:attribute>
    <xsl:value-of select="@nombre"/>
  </xsl:element>
</tr>
</xsl:template>

```

Ordena las pizzas por precio

Calcula la media de los precios

Genera referencias de la forma
...

```
<ns>
<num>5</num>
<num>6</num>
<num>7</num>
<num>8</num>
<num>9</num>
<num>10</num>
<num>11</num>
<num>12</num>
<num>100</num>
</ns>
```



Factorial

- 5! = 120
- 6! = 720
- 7! = 5040
- 8! = 40320
- 9! = 362880
- 10! = 3628800
- 11! = 39916800
- 12! = 479001600
- 100! = NaN

```
<xsl:template match="num">
  <li>
    <xsl:value-of select="."/> =
    <xsl:call-template name="fact">
      <xsl:with-param name="x"><xsl:value-of select="." />
    </xsl:with-param>
    </xsl:call-template>
  </li>
</xsl:template>

<xsl:template name="fact">
  <xsl:param name="x" />
  <xsl:choose> <xsl:when test="$x = 0">1</xsl:when>
  <xsl:otherwise>
    <xsl:variable name="llamada">
      <xsl:call-template name="fact">
        <xsl:with-param name="x"><xsl:value-of select="$x - 1" />
      </xsl:with-param>
    </xsl:call-template>
    </xsl:variable>
    <xsl:value-of select="$llamada * $x" />
  </xsl:otherwise>
</xsl:choose>
</xsl:template>
```

fact x = if x = 0 then 1
else x * fact (x - 1)

- Es un lenguaje de programación Turing-completo
 - Lenguaje declarativo (sin asignación destructiva)
 - Admite recursividad, funciones de orden superior, evaluación perezosa
"The functional programming XSLT – a proof through examples" (D. Novatchev)
<http://www.xml.top/xsl/articles/fp>
- 4 tipos de datos (enteros, booleanos, strings y conjuntos de nodos)
 - Sin chequeo estático de tipos
 - Seguridad? Eficiencia? (no son objetivos de diseño)
- Flexible: La sintaxis de XPath se adapta a posibles cambios en la estructura. No se valida el documento
 - Bueno para hacer tareas sencillas rápidamente
 - Empotrado en navegadores
- Los programas XSLT son documentos XML
 - La sintaxis es poco amigable para el programador
 - Necesidad de *buenas* herramientas de autor

Vocabularios específicos 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

Discusión sobre XML: Ventajas

- 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
- 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 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>
```

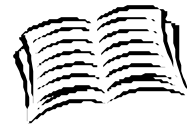
- Es posible crear formatos y vocabularios propietarios
- Puede fomentar la proliferación de vocabularios específicos
- Bueno para texto. malo para datos binarios

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

- Poco eficiente como lenguaje de almacenamiento de bases de datos

Selección de Enlaces

- Página del consorcio: <http://www.w3c.org>
 En español: <http://www.it.uc3m.es/~xml/enlaces.html>
 Especificación anotada: <http://www.xml.com/axml/testaxml.htm>
 XML en industria: <http://www.xml.org>
 Diseño de vocabularios XML: <http://www.xmlpatterns.com>
 Tutoriales: <http://www.w3schools.com>
 Artículos de XML:
 <http://www.topxml.com>
 <http://www.xmlpatterns.com>
 Software de XML
 1. <http://www.xmlsoftware.com>
 2. <http://www.xmlhack.com>
 3. <http://www.garshol.priv.no/download/xmltools/>



Fin de la Presentación

