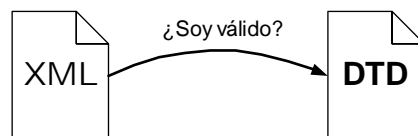


## 3.1 Tipos de DTDs

### Introducción (I)

- El DTD es un documento que nos permite definir un dialecto XML.
  - DTD → Document Type Definition
- Si queremos validar un documento XML (para comprobar si cumple las normas de un dialecto) tendremos que "validarlo contra el DTD".





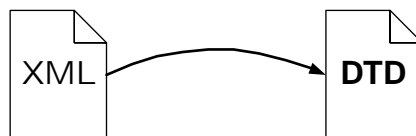
## Introducción (II)

- El DTD contiene una serie de declaraciones (especificamos *formalmente* qué elementos tiene el dialectos, qué atributos, etc.)
- Tiene una sintaxis concreta.
- Un DTD puede ser:
  - Externo
  - Interno
  - Publico



## DTD Externo (I)

- DTD Externo: El DTD se encuentra en un documento separado del documento XML.
- Para validar el documento XML hay que acceder a ese documento DTD.





## DTD Externo (II)

- ¡Habr  que 'decirle' al documento XML donde est  ese DTD!
- Utilizamos la declaraci n `<!DOCTYPE>`
- Tiene que aparecer en el documento XML *antes* del elemento ra z.
- En el caso de un DTD externo, la sintaxis es:

```
<!DOCTYPE raiz SYSTEM "URL">
```



## DTD Externo (III)

- raiz → Nombre del elemento ra z del documento.
- URL → URL del DTD. Puede ser *cualquier* URL v lida.

```
<!DOCTYPE listas SYSTEM "listaml.dtd">
```

El elemento ra z del documento es `listas`. El DTD del dialecto XML al que se tiene que ajustar el documento XML es `listaml.dtd` (es una URL, por lo que el documento DTD `listaml.dtd` est  en el mismo directorio que el documento XML)



## DTD Externo (IV)

```
<?xml version="1.0"?>
<!DOCTYPE listas SYSTEM "listaml.dtd">
<listas>
  <lista nombre="amigos" desc="Mis amigos">
    <subs mail="amigo1@hotmail.com">
    <subs mail="amigo2@miexmail.com">
  </lista>
</listas>
```



## DTD Externo (V)

- Ventaja del DTD Externo: Varios documentos XML pueden validarse contra un único documento DTD.
- Desventaja: Hay que acceder a un documento separado para la validación. Este documento puede no estar 'cerca' del documento XML.  
P.ej.:

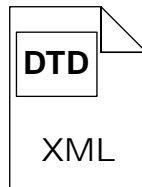
```
<!DOCTYPE listas SYSTEM "http://www.dtd.com/listaml.dtd">
```

¡Tengo que ir hasta [www.dtd.com](http://www.dtd.com) para hacer la validación!



## DTD Interno (I)

- DTD Interno: El DTD se encuentra en el propio documento XML.
- Para realizar la validación no hay que acudir a ningún documento separado.



## DTD Interno (II)

- De nuevo, utilizamos la declaración `<!DOCTYPE>`
- En el caso de los DTDs internos, la sintaxis es:  

```
<!DOCTYPE raiz [  
    declaraciones DTD  

```



## DTD Interno (III)

- raíz → Nombre del elemento raíz del documento.
- Las declaraciones DTD (elementos, etc.) van entre los corchetes. Esto mismo podríamos ponerlo en un documento externo (DTD externo)



## DTD Interno (IV)

- Ventaja: Tenemos el DTD en el propio documento. No hay que acceder a documentos externos.
- Desventaja: Ese DTD interno no puede ser accedido por otros documentos XML. Habría que repetir el DTD en cada documento.



## DTD Interno (V)

- No es normal utilizar exclusivamente DTDs internos. Lo mas normal es combinar DTDs internos y externos.
- El DTD interno tiene prioridad. Nos permitirá *redefinir* partes del DTD externo.

```
<!DOCTYPE raiz SYSTEM URL [  
    declaraciones internas DTD  
>
```



## DTD Público (I)

- DTD Publico: Cuando un DTD goza de la suficiente aceptación se le puede asignar un *identificador público*.
  - Identifica el DTD con una sintaxis especifica, y sin proporcionar su URL.
  - Algunos programas pueden reconocer ese identificador público. Es posible que ya tenga almacenado el DTD y no tenga que acudir a Internet para obtenerlo. Por ejemplo, se prevé que todos los navegadores de Internet incluyan en el futuro todos los DTDs de XHTML. De esa manera, para validar un documento XHTML no hay que acudir hasta la web del W3C para conseguir el DTD.



## DTD Público (II)

- Sintaxis de un DTD Público:

```
<!DOCTYPE raiz PUBLIC "ID Publico" "URL">
```

- *raiz*: Nombre del elemento raíz del documento.
- *ID Publico*: Identificador publico del DTD.
- *URL*: URL donde puede encontrarse el DTD. Se proporciona por si el programa no dispone del DTD correspondiente al identificador publico especificado.



## DTD Público (III)

- Ejemplo. Un documento XHTML:

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

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

```
<html>  
  <head>  
    <title>Ejemplo XHTML</title>  
  </head>  
  <body>  
    Hol a mundo!  
  </body>  
</html>
```





## 3.2 Declaraciones en DTDs

---



### Declaraciones (I)

---

- Un DTD (externo, interno, o público) consiste de declaraciones.
  - De elementos: ¿Qué elementos (“etiquetas”) tiene mi dialecto XML?
  - De atributos: ¿Qué atributos tendrán los elementos de mi dialecto XML?
  - De entidades: ¿Qué entidades tendrá mi dialecto XML?



## Declaraciones (II)

---

- Con estas tres declaraciones podemos definir por completo un dialecto XML.
  - Elementos: `<!ELEMENT>`
  - Atributos: `<!ATTLIST>`
  - Entidades: `<!ENTITY>`

## 3.3 Declaraciones de Elementos

### Declaraciones de Elementos (I)

- Un DTD debe incluir tantas declaraciones de elementos como tipos de elementos ("tipos de etiqueta").
- Al declarar un elemento tendremos que indicar:
  - El nombre del elemento ("nombre de la etiqueta")
  - Los posibles contenidos del elemento (¿texto? ¿otros elementos? ¿qué elementos? ¿contenido mixto?)



## Declaraciones de Elementos (II)

---

- La sintaxis de una declaración de elemento es:  
  
`<! ELEMENT nombre (contenido) >`
  - `nombre` → el nombre del elemento
  - `contenido` → ¿qué puede contener este elemento?



## Contenido: Texto

---

- Texto: Lo denotamos con la palabra clave `#PCDATA`  
`<! ELEMENT strong (#PCDATA) >`  
Declaramos un elemento `strong` (`<strong>...</strong>`) que sólo puede contener texto.



## Contenido: Un Elemento (I)

- Un elemento: Lo denotamos indicando el nombre del elemento.

`<! ELEMENT html (body) >`

Declaramos un elemento html (`<html>...</html>`) que sólo puede contener un elemento body.

- ¡Ojo!
  - También habrá que declarar el elemento body.
  - En esta declaración, dentro de un elemento html sólo puede haber un elemento body. No puede haber texto, más de un elemento body, otros elementos, etc.



## Contenido: Un Elemento (II)

```
<html >
  <body>
    ...
  </body>
</html >
```




BIEN. Dentro del elemento html sólo hay un único elemento body.

```
<html >
  <head>... </head>
  <body>
    ...
  </body>
</html >
```




MAL. Este documento sería válido en HTML, pero no es válido si lo validamos con nuestro DTD (el elemento html únicamente puede contener un elemento body, no un elemento head)



## Contenido: Más de un Elemento (I)

- Más de un elemento: Podemos controlar el número de elementos añadiendo un sufijo al nombre del elemento:
  - \* → 0 o más elementos.
  - + → 1 o más elementos.
  - ? → 0 o 1 elemento.



## Contenido: Más de un Elemento (II)


- Por ejemplo:  
`<! ELEMENT body (p*) >`  
Declaramos un elemento body (`<body>...</body>`) que puede contener cero o más elementos p.
- ¡Ojo!
  - El DTD es muy estricto. Seguimos sin poder añadir texto u otros elementos.

## Contenido: Más de un Elemento (III)

<pre>&lt;body&gt;   &lt;p&gt;... &lt;/p&gt;   &lt;p&gt;... &lt;/p&gt; &lt;/body&gt;</pre>	→	BIEN. Dentro del elemento body hay cero o más (en este caso, 2) elementos p.
<pre>&lt;body&gt;   Voy a poner un   &lt;p&gt;el elemento p&lt;/p&gt;   &lt;p&gt;y otro&lt;/p&gt; &lt;/body&gt;</pre>	→	MAL. En nuestro DTD, el elemento body <u>sólo</u> puede contener elementos p (cero o más). En este documento, body contiene elementos p y texto (contenido mixto).

## Contenido: Secuencias de Elementos (I)

- Secuencias de elementos: Podemos especificar que el contenido son varios elementos *distintos* en un orden concreto.
- Por ejemplo, el contenido de un elemento "ordenador" podría ser un elemento "procesador" seguido de un elemento "memoria".



## Contenido: Secuencias de Elementos (II)

```
<ordenador>  
  <procesador>Pentium III</procesador>  
  <memoria>128 MB</memoria>  
</ordenador>
```

- En el DTD haremos esto de la siguiente manera:


```
<!ELEMENT ordenador (procesador, memoria)>  
<!ELEMENT procesador (#PCDATA)>  
<!ELEMENT memoria (#PCDATA)>
```



## Contenido: Secuencias de Elementos (III)


- Las secuencias de elementos se separan por comas.
- El orden es *significativo*
  - No sería válido poner primero memoria y luego procesador.
- A los elementos podemos añadirles los sufijos \*, +, ó ? para modificar la cantidad permitida de elementos.





## Contenido: Selecciones de Elementos (I)

- Selecciones: Podemos especificar que el contenido deber ser un (y solo un) elemento de varios posibles.
- Por ejemplo, un elemento "producto" (un producto en una tienda) puede contener o un elemento "pesetas" o un elemento "euros" (pero nunca los dos a la vez).




## Contenido: Selecciones de Elementos (II)

```
<producto>  
  <pesetas>10000</pesetas>  
</producto>  
<producto>  
  <euros>15000</euros>  
</producto>
```

- En el DTD haremos esto de la siguiente manera:

```
<! ELEMENT producto (pesetas | euros)>  
<! ELEMENT pesetas (#PCDATA)>  
<! ELEMENT euros(#PCDATA)>
```



## Contenido: Selecciones de Elementos (III)

- Las selecciones de elementos (o “disyunciones”) van separadas por el carácter de la barra vertical → |
- Podemos especificar tantos posibles elementos como queramos.  
`<! ELEMENT producto (pesetas|euros|dol ares|l i bras)>`
- Sin embargo, el contenido del elemento (producto) solo podrá ser uno de los elementos especificados en la selección.



## Contenido: Combinación de todo lo anterior (I)

- Podemos combinar secuencias, selecciones, y sufijos para especificar prácticamente cualquier tipo de contenido.
- Casi siempre será necesario utilizar paréntesis.



## Contenido: Combinación de todo lo anterior (II)

- Ejemplos:

- $(a,(b|c)) \rightarrow$  Un elemento "a" seguido de o un elemento "b" o un elemento "c".
- $((a,b) | (c,d)) \rightarrow$  O un elemento "a" seguido de un elemento "b" o un elemento "c" seguido de un elemento "d".
- $(x | (a,b+)) \rightarrow$  O un elemento "x" o un elemento "a" seguido de (por lo menos) un elemento "b".



## Contenido: Contenido Mixto

- Contenido Mixto. Se declara de la siguiente manera:

```
<! ELEMENT body ( ( #PCDATA | b | i ) * ) >
```

Declaramos un elemento body con contenido mixto. Además de texto, podemos incluir tantos elementos "b" e "i" como queramos.



## Contenido: EMPTY

- Elementos *empty* o singulares. No contienen nada. Se declaran de la siguiente manera:

```
<! ELEMENT i mg EMPTY>
```



## Contenido: ANY (I)

- Elementos ANY. Pueden contener *cualquier elemento o texto*. Se recomienda utilizarlos únicamente cuando estamos haciendo pruebas (nunca en la versión definitiva de un DTD). Se declaran de la siguiente manera:

```
<! ELEMENT vaso ANY>
```



## Contenido: ANY (II)

---

- ¡Ojo! Los elementos contenidos dentro de un elemento ANY tienen que ser coherentes con el resto del DTD.

## 3.4 Declaraciones de Atributos



---

### Declaraciones de Atributos (I)



---

- Una vez que hemos declarado los elementos de nuestro lenguaje, seguramente nos interesará declarar *atributos* para los elementos.
- La haremos con la declaración `<!ATTLIST>`.
- Con esta declaración podremos especificar *qué atributos* tiene un elemento.



## Declaraciones de Atributos (II)

- Sintaxis:

`<! ATTLIST elemento atributos>`

- `elemento` → nombre del elemento cuyos atributos vamos a especificar.
- `atributos` → lista de los atributos del elemento.



## Declaraciones de Atributos (III)

- Los atributos del elemento tienen la siguiente sintaxis:

`nombre tipo valor_por_defecto`

- Si tenemos varios atributos:

`nombre1 tipo1 valor_por_defecto1`

`nombre2 tipo2 valor_por_defecto2`

`nombre3 tipo3 valor_por_defecto3`



## Declaraciones de Atributos (IV)

- Por ejemplo, un elemento ordenador con tres atributos (procesador, velocidad, y memoria)

```
<!ELEMENT ordenador EMPTY>
```

```
<!--ATTLIST ordenador
```

```
    procesador CDATA #REQUIRED
```

```
    velocidad CDATA #REQUIRED
```

```
    memoria CDATA #REQUIRED
```

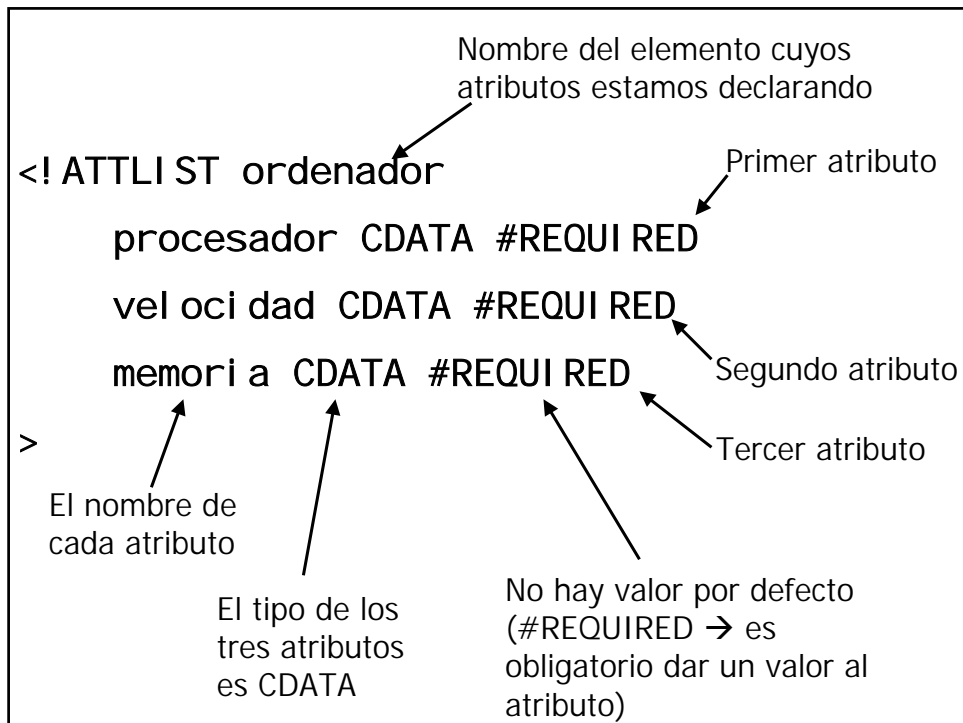
```
>
```



## Declaraciones de Atributos (V)

- Primero declaramos el elemento (<!ELEMENT>).
- Una vez declarado, declaramos todos sus atributos en una única declaración <!--ATTLIST>.





## Nombre del Atributo

- El nombre de un atributo debe ser un *nombre XML*.
- No puede haber dos atributos con el mismo nombre en un mismo elemento.



## Tipos de Atributos (I)

- Existen 10 tipos de atributos:
  - CDATA
  - NMTOKEN
  - NMTOKENS
  - Enumeración
  - ENTITY
  - ENTITIES
  - ID
  - IDREF
  - IDREFS
  - NOTATION



## Tipos de Atributos (II)

- CDATA: El valor del atributo debe ser texto (cualquier cadena de caracteres).
- Es un tipo de datos muy general. No distingue entre texto, numero, booleanos, porcentajes, etc.



## Tipos de Atributos (III)

- NMTOKEN: Parecido a CDATA. Sin embargo, el valor del atributo debe ser un *name token*.
- Name token → Igual que un *nombre XML*. Sin embargo, en un name token no hay ninguna restricción en el primer carácter (en un nombre XML era obligatorio utilizar un carácter alfabético o el carácter de subrayado).
  - 3dimension → Es un name token válido. No es un nombre XML válido.
  - alt → Name token y nombre XML válido.



## Tipos de Atributos (IV)

- NMTOKENS: El valor del atributo deben ser varios name tokens separados por espacios.

```
<! ELEMENT ti pofi ch EMPTY>  
<! ATTLIST ti pofi ch  
    ext NMTOKENS #REQUIRED  
>
```

```
<ti pofi ch ext=".htm .html .asp .php" />
```



## Tipos de Atributos (V)

- Enumeración: No es un 'tipo de datos' (no ponemos la palabra clave ENUMERATION). Indicamos, en lugar del tipo de datos, todos los posibles valores del atributo (entre paréntesis y separados por el carácter de la barra vertical).
- Los posibles valores deben ser name tokens.



## Tipos de Atributos (VI)

```
<!ELEMENT texto (#PCDATA)>  
<!ATTLIST texto  
  al i neaci on (i zq | centr | der) #REQUI RED  
>
```

```
<texto al i neaci on="i zq" >  
Este texto debería alinearse a l a i zqui erda.  
</texto>
```



## Tipos de Atributos (VII)

- ID: El valor del atributo debe ser un nombre XML (*no* un name token) que identifique de manera única al elemento en el documento.
- Es algo así como decir que el atributo va a especificar la 'clave primaria' del elemento.



## Tipos de Atributos (VIII)

```
<! ELEMENT empl eado EMPTY>  
<! ATTLIST empl eado  
    dni ID #REQUIRED  
>
```

```
<empl eado dni = "_17288890" />  
<empl eado dni = "_17467690" />  
<empl eado dni = "_17264345" />  
<empl eado dni = "_45448890" />
```



## Tipos de Atributos (IX)

- ¡Ojo! El valor debe ser un *nombre XML*, por lo que los números no son valores válidos para un atributo de tipo ID.



## Tipos de Atributos (X)

- IDREF: El atributo debe tomar el valor que tome algún atributo de tipo ID de cualquier elemento en el documento.  
(IDREF → REFerencia a ID)
- Es algo así como especificar una 'clave extranjera' en un elemento.



## Tipos de Atributos (XI)

```
<! ELEMENT empl eado EMPTY>
<! ATTLIST empl eado
  dni ID #REQUIRED
  dep IDREF #REQUIRED
>

<! ELEMENT departamento EMPTY>
<! ATTLIST departamento
  cod ID #REQUIRED
  nombre CDATA #REQUIRED
>
```



## Tipos de Atributos (XII)

```
<departamento cod="Inf" nombre="Informati ca" />
<departamento cod="Con" nombre="Contabi l i dad" />

<empl eado dni =" _17288890" dep=" Inf" />
<empl eado dni =" _17467690" dep=" Con" />
<empl eado dni =" _17264345" dep=" Inf" />
<empl eado dni =" _45448890" dep=" Inf" />
```



## Tipos de Atributos (XIII)

- IDREFS: El valor del atributo deben ser varios IDREF (referencias a ID) separados por espacios.



## Valores por Defecto (I)

- #REQUIRED: Es obligatorio dar un valor al atributo. No especificamos un valor por defecto.

`dni ID #REQUIRED`

- #IMPLIED: Dar un valor al atributo es opcional.

`estado_civil CDATA #IMPLIED`





## Valores por Defecto (II)

- **#FIXED *valor***: El valor del atributo es constante e inmutable (y es igual a *valor*).

empresa CDATA #FIXED "Mi empresa"

- ***valor***: Si no se le da un valor explícitamente al atributo, se da por supuesto que es *valor*.

visible CDATA "TRUE"

## 3.5 Declaraciones de Entidad

---

### Declaraciones de Entidad (I)

---

- En XML vienen definidas por defecto 5 entidades:
    - **lt** → Less Than (menor que)
    - **amp** → Ampersand
    - **gt** → Greater than (mayor que)
    - **quot** → Quotation Mark (comilla doble)
    - **apos** → Apostrophe (comilla simple)
- (las referenciamos escribiendo &nombre\_entidad; → Referencia a Entidad)



## Declaraciones de Entidad (II)

- En el DTD podemos definir nuestras propias entidades (para luego referenciarlas) utilizando la declaración `<!ENTITY>`

`<! ENTITY nombre contenido>`

- `nombre` → nombre de la entidad
- `contenido` → Texto que es insertado en el documento cuando referenciamos la entidad.



## Declaraciones de Entidad (III)

- Existen 4 tipos de entidades:
  - Entidades generales
  - Entidades generales externas parseadas
  - Entidades externas no parseadas
  - Entidades paramétricas



## Entidades Generales (I)

- Las entidades generales nos permiten especificar directamente el texto por el que debe sustituirse la referencia a la entidad.

`<! ENTITY nombre " texto" >`



## Entidades Generales (II)

### EJEMPLO

En el DTD:

`<! ENTITY adn " Áci do Desoxi rri bonucl ei co" >`

En el documento XML:

El **&adn;** es fundamental para la existencia de la vida en el planeta Tierra. Sin **&adn;** no habría seres vivos en la Tierra. ¡ Como me gusta el **&adn;** !



## Entidades Generales (III)

Resultado:

El **Ácido Desoxirribonucleico** es fundamental para la existencia de la vida en el planeta Tierra. Sin **Ácido Desoxirribonucleico** no habría seres vivos en la Tierra. ¡Como me gusta el **Ácido Desoxirribonucleico**!



## Entidades Generales (IV)

- El contenido de la entidad debe estar bien formado.
- Estas entidades pueden referenciarse desde cualquier lugar del documento XML. No pueden referenciarse dentro del DTD.



## Entidades Externas (I)

- Estas entidades nos permiten especificar un *fichero* que será insertado en el lugar de la referencia a la entidad.

<! ENTITY *nombre* SYSTEM " URL" >

- URL → URL del fichero que será insertado en el documento cuando se referencia la entidad.



## Entidades Externas (II)

- No podemos referenciar estas entidades *dentro del valor de un atributo*.
- Este tipo de entidad realiza la misma labor que un <!--#include--> de los SSI, pero utilizando sintaxis XML.



## Entidades Parametricas (I)

- Estas entidades nos permiten utilizar referencias a entidades dentro de un DTD. Se declaran de manera distinta:

```
<! ENTITY % nombre " texto" >
```

También se referencian de manera distinta:

```
%nombre_entidad;
```



## Entidades Parametricas (II)

- ¿Para que sirven? Nos van a permitir evitar declaraciones del siguiente tipo:

```
<! ELEMENT coche (ruedas, col or, puertas)>
```

```
<! ELEMENT cami on (ruedas, col or, potenci a)>
```

```
<! ELEMENT moto (ruedas, col or, cc)>
```

- ¡"ruedas, col or" se repite en todas las declaraciones!



## Entidades Parametricas (III)

- Podemos declarar una entidad paramétrica:

```
<! ENTITY % partes_vehiculo "ruedas, color" >
```

- El DTD anterior se nos queda en:

```
<! ELEMENT coche (%partes_vehiculo; , puertas) >  
<! ELEMENT camion (%partes_vehiculo; , potencia) >  
<! ELEMENT moto (%partes_vehiculo; , cc) >
```





## 3.6 XML Schema

---



### XML Schema (I)

---

- El DTD es una herramienta de especificación de lenguajes muy limitada.
- Nos permite especificar toda la estructura del lenguaje, pero deja en el aire muchas cosas, especialmente los tipos de datos de los atributos (con el DTD, todo es CDATA → texto)



## XML Schema (II)

- Para superar esta limitación, la W3C ha desarrollado un dialecto XML conocido como "XML Schema". Con este dialecto podemos definir con rigor todos los detalles de un dialecto XML.
- Alcanzó el estatus de recomendación de la W3C en Mayo de 2001.



## XML Schema (III)

- Es más difícil escribir un Schema que un DTD, porque hay que tener más cosas en cuenta (tipos de datos, restricciones, etc.)
- Sin embargo, un Schema nos proporciona mucha más información sobre el dialecto que un DTD.



## XML Schema (IV)

---

- Está especialmente indicado para:
  - Bases de datos relacionales
  - Sistemas con un diseño OO
  
- Más información:

<http://www.w3.org/XML/Schema>