Arquitecturas Distribuidas

Tema 3. IV. XML

IV. XML

1. ¿Qué es XML?
2. Estructura de un documento XML
3. Importancia de XML
4. Problemas de XML
5. Conformidad y validez
6. Estructura DTD XML
   - 3.1. Elementos
   - 3.2. Atributos
   - 3.3. Notaciones
   - 3.4. Entidades
   - 3.5. Instrucciones de procesado (PI)
7. Interpretación de un documento XML por un navegador
8. XHTML
9. Tecnologías relacionadas con XML
   - 5.1. Espacios de nombres
   - 5.2. XSchema
   - 5.3. Lenguajes de estilo y transformación para XML

¿Qué es XML? (I)

- Lenguaje de Marcas eXtendido.
- **Meta-lenguaje** para definir lenguajes de marcas particulares: permite crear otros lenguajes de marcas
  - Proporciona las reglas sintácticas que permiten crear lenguajes que están **bien formados** y puedan ser **validados**
  - Es decir, permite declarar el DTD de un lenguaje de marcas particular
  - Cuando alguien crea un documento XML, define sus propias etiquetas. No está limitado en qué o cómo se describen los datos.
  - Mientras el documento esté bien formado es XML aceptable
- Cada lenguaje particular (e.g. XHTML, MathML, SVG) es una aplicación de XML

¿Qué es XML? (II)

- Separa los datos de su representación (formato) particular
- XML no es HTML mejorado, sino SGML simplificado.
  - Proporciona reglas sintácticas para crear lenguajes que deben estar bien formados y pueden ser validados
- Las marcas proporcionan información acerca de los datos (metadatos)
  - XML permite también especificar **la semántica** de los datos que contiene: ¿qué significan? Esto es útil en ciertos ámbitos.
  - En otros ámbitos es preferible sólo especificar la estructura, como HTML, que utiliza marcas para definir la estructura de un documento no la semántica de los datos que contiene: es más general.
- Separación de los datos de la forma en que se procesan = independiente del dominio de aplicación y adecuado para sistemas distribuidos
  - Legible por los humanos *(Human-readable)*: ¿por qué? al menos hay una oportunidad de discernir el significado en el futuro

Estructura de un documento XML

- Prólogo (declaraciones) y cuerpo.

```
<?xml version="1.0" [parámetros opcionales]>
Declaraciones de las reglas del documento

<raiz> Contenido </raiz>
```

PRÓLOGO

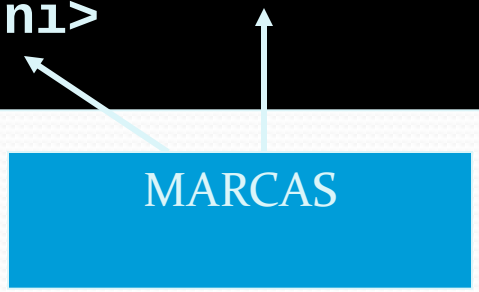
CUERPO

- Parámetros opcionales en la declaración:
  - `standalone`: indica si el documento es autocontenido.
  - `encoding`: juego de caracteres del documento.

Estructura de un documento XML

- Ejemplo documento XML simple:

```
<?xml version="1.0" encoding='UTF-8‘?>

<dni>
        <id> 568767331X </id>
        <nombre> Pepe </nombre>
        <apellidos> Pérez </apellidos>
        <nacimiento>
                <lugar> Murcia </lugar>
                <fecha> 1/1/1983 </fecha>
        </nacimiento>
</dni>
```



Estructura de un documento XML

- Ejemplo documento XML simple:

```
<?xml version=

<dni>
      <id> 568
      <nombre>
      <apellid
      <nacimie
            <l                          gar>
            <f
      </nacimiento>
</dni>
```

Una aplicación podría usar el DNI para enviar a la máquina que los imprime los comandos oportunos

Estructura de un documento XML

- Ejemplo documento XML simple:

```
<?xml version=
<dni>
        <id> 568
        <nombre>
        <apellid
        <nacimie
                                                gar>
<
```

Una aplicación podría usar el DNI para enviar a la máquina que los imprime los comandos

Otra aplicación podría usar el DNI para enviar al usuario un correo para recordarle que debe renovarlo.

Estructura de un documento XML

- Ejemplo documento XML simple:

```
<?xml version=
<dni>
        <id> 568
        <nombre>
        <apellid
        <nacimie
                                    pereiil </lugar>
<
```

Una aplicación podría usar el DNI para enviar a la máquina que los hace los comandos oportunos

Otra aplicación podría usar el DNI para enviar al usuario un correo para recordarle que debe renovarlo.

Otra aplicación podría usar los datos para contrastar la veracidad en una compra a través de Internet.

Estructura de un documento XML

- XML PROPORCIONA UN MÉTODO PARA ESTRUCTURAR INFORMACIÓN, INDEPENDIENTEMENTE DE LA APLICACIÓN QUE LA PROCESE Y PARA DOTARLA DE SIGNIFICADO

Importancia de XML

- Estándar y MUY utilizado. Es usado por muchas otras tecnologías y en muchos ámbitos:
  - los ficheros de configuración de las aplicaciones de Microsoft son XML.
  - KML es el lenguaje utilizado por Google Maps y Google Earth
  - SVG, scalable vector graphics
  - Office Open XML, usador por las nuevas versiones de Microsof Office
  - Cientos de ejemplos más

Problemas de XML

- Tremendamente redundante (desde el punto de vista de una máquina). Muy poco eficiente en el uso de recursos
  - Ancho de banda, latencia, almacenamiento.
- Proliferación de lenguajes
  - Necesitamos una aplicación específica para cada uno de ellos
  - El parser es general (por ejemplo la interfaz DOM), sin embargo cada lenguaje necesita una aplicación que interprete adecuadamente las marcas.

Conformidad y validez

- En un documento bien formado (*well-formed*)
  - *Se cierran todos los elementos abiertos*

```
HTML: <h2> <b> Nombre: </b> <h3> <i> Pepe Pérez </i>

XML: <h2> <b> Nombre: </b> </h2> <h3> <i> Pepe Pérez </i> </h3>
```

Conformidad y validez

- En un documento bien formado (*well-formed*)
  - *Se identifican los elementos vacíos*

```
HTML: <b> Nombre: </b> <i> Pepe Pérez </i> <br>

XML: <b> Nombre: </b> <i> Pepe Pérez </i> <br/>
```

Elemento vacío: <TAG/>

Conformidad y validez

- En un documento bien formado (*well-formed*)
  - *Los valores de los atributos siempre están presentes y van entre comillas (“valor” o ‘valor’)*

```
HTML: <li compact>
      <li type=disc>

XML: <li compact="compact“>
     <li type="disc">
```

Conformidad y validez

- En un documento bien formado (*well-formed*)
  - *En elementos y atributos, las mayúsculas y minúsculas son letras distintas*

```
HTML: <LI type=disc> == <li type=disc> == <Li tYPe=disc>

XML: <LI type='disc'> != <li type='disc'> != <Li tYPe='disc'>
```

Conformidad y validez

- En un documento bien formado (*well-formed*)
  - *Los elementos deben estar bien anidados*

HTML: El analizador (parser) intenta arreglarlo

```
HTML: <b> <u> Número de DNI: 568767331X </b> </u>

XML: <b> <u> Número de DNI: 568767331X </b> </u>
```

XML: El analizador (parser) debe dar error!

Conformidad y validez

- En un documento bien formado (*well-formed*)
  - *Todas las entidades deben estar declaradas*

```
HTML: <b> Nombre   Apellidos </b>

XML: <b> Nombre   Apellidos </b>
```

- XML: necesita una declaración previa.
- XML sólo entiende por defecto:

  & (&), < (<), > (>), &quote; (“) y ' (‘)

Conformidad y validez

- En un documento bien formado (*well-formed*)
  - *Puede contener comentarios*

```
XML: <!-- Comentario -->
```

  - *Puede contener datos que no se analicen sintácticamente*

```
XML: <![CDATA [datos que no se analizan]]>
```

Conformidad y validez

- XML admite una declaración de Definición del Tipo de Documento (DTD):

  "Conjunto que describe de reglas que debe respetar un documento perteneciente a un lenguaje XML específico. Define la estructura de un documento: elementos, contenidos de cada elemento, orden, atributos de cada elemento, etc."

- Un documento es conforme ó valido si y sólo si:
  - **Está bien formado**
  - **Tiene un DTD y respeta sus reglas**

Estructura DTD XML

- Declaración del DTD:

```
<?xml version="1.0">
<!DOCTYPE nombre [
        REGLAS SINTACTICAS (declaraciones internas)
]>

<nombre>
...
</nombre>
```

Estructura DTD XML

- Declaración “externa”:

```
<?xml version="1.0">
<!DOCTYPE nombre (SYSTEM|PUBLIC) URL [ ]>

<nombre>
...
</nombre>
```

URL donde se ubica el documento con el DTD.

SYSTEM si es un DTD propio

PUBLIC si es un DTD estándar.

Estructura DTD XML

- Declaración “mixta”:

```
<?xml version="1.0">
<!DOCTYPE nombre (SYSTEM|PUBLIC) URL [

        Reglas sintácticas (declaraciones internas)

 ]>

<nombre>
...
</nombre>
```

DTD es la suma de las partes externas e internas

Estructura DTD XML

- Observación:

```
<?xml version="1.0">
<!DOCTYPE nombre [
        REGLAS SINTACTICAS (declaraciones internas)
]>

<nombre>
...
</nombre>
```

Nombre indica cómo se llama la raíz del documento

Estructura DTD XML

- Reglas del DTD:

```
<!DOCTYPE nombre [

REGLAS SINTACTICAS:

        ELEMENTOS:      <!ELEMENT elemento (contenido)>
        ATRIBUTOS:      <!ATTLIST elemento
                                atributo tipo descripción
                                ...
                        >
        ENTIDADES:
                INTERNAS: <!ENTITY [%] entidad “cualquier texto”>
                EXTERNAS: <!ENTITY [%] entidad URL>
                PREDEFINIDAS: &amp; &lt; etc

        NOTACIONES:     <!NOTATION nombre identificador>

        INSTRUCCIONES: <?... >
]
```

- Utiliza la sintaxis de SGML, simplificando las posibilidades

Elementos en DTD XML

- Declaración de los elementos:

```
<!ELEMENT elemento (contenido)>
```

Nombre del nuevo elemento

Declaración del tipo de contenido (elementos y datos) que puede incluir el elemento, cuántos y en qué orden aparecen.

Indica la declaración de un nuevo elemento

Elementos en DTD XML

Clases de contenido para elementos (5 posibilidades):

```
<!--Elemento vacío-->
<!ELEMENT elemento EMPTY>

<!--Puede contener cualquier elemento-->
<!ELEMENT elemento ALL>

<!--Contenido es una cadena de caracteres (datos) con entidades (que deben ser procesadas)-->
<!ELEMENT elemento (#PCDATA)>

<!--Elemento compuesto de otros elementos, según una fórmula-->
<!ELEMENT elemento formula(elementos)>

<!--Elemento con contenido mixto (otros elementos y #PCDATA)-->
<!ELEMENT elemento formula(elementos, #PCDATA)>
```

Elementos en DTD XML

Clases de contenido para elementos (fórmulas):

- Sintáxis muy similar a SGML.
- Indican el número de veces que puede aparecer un componente (cardinalidad):

  * *(cero o más veces)*

  \+ *(una o más veces)*

  ? *(cero o una vez)*
- Indican el orden en que deben/pueden aparecer los elementos (atención: el operador **&** no existe, diferencia con SGML):

  a,b *(b a continuación de a)*

  a*|*b *(b ó a)*

  (conjunto) *(agrupación, si existe operador posterior afecta a todo el conjunto)*

Elementos en DTD XML

Ejemplos de fórmulas:

```
<!ELEMENT a (b,c)>

        <a>
                <b>contenido de b</b>
                <c>contenido de c</c>
        </a>


        <a>
                <c>contenido de c</c>
                <b>contenido de b</b>
        </a>


        <a>
                <b>contenido de b</b>
        </a>

        <a></a>
```

Elementos en DTD XML

Ejemplos de fórmulas:

```
<!ELEMENT a (b,c)>

        <a>
                <b>contenido de b</b>
                <c>contenido de c</c>
        </a>

        <a>
                <c>contenido de c</c>
                <b>contenido de b</b>
        </a>

        <a>
                <b>contenido de
        </a>

        <a> </a>
```

VÁLIDA!

NO VÁLIDA!

NO VÁLIDA!

NO VÁLIDA!

Elementos en DTD XML

Ejemplos de fórmulas:

```
<!ELEMENT a (b?, c)+>

        <a>
                <c>contenido de c</c>
        </a>



        <a>
                <b>contenido de b</b>
                <c>contenido de c</c>
                <c>contenido de c</c>
                <b>contenido de b</b>
                <c>contenido de c</c>
        </a>
```

Elementos en DTD XML

Ejemplos de fórmulas:

```
<!ELEMENT a (b?, c)+>

        <a>
                <c>contenido de c</c>
        </a>

        <a>
                <b>contenido de b</b>
                <c>contenido de c</c>
                <c>contenido de
                <b>contenido de
                <c>contenido de
        </a>
```

VÁLIDA!

VÁLIDA!

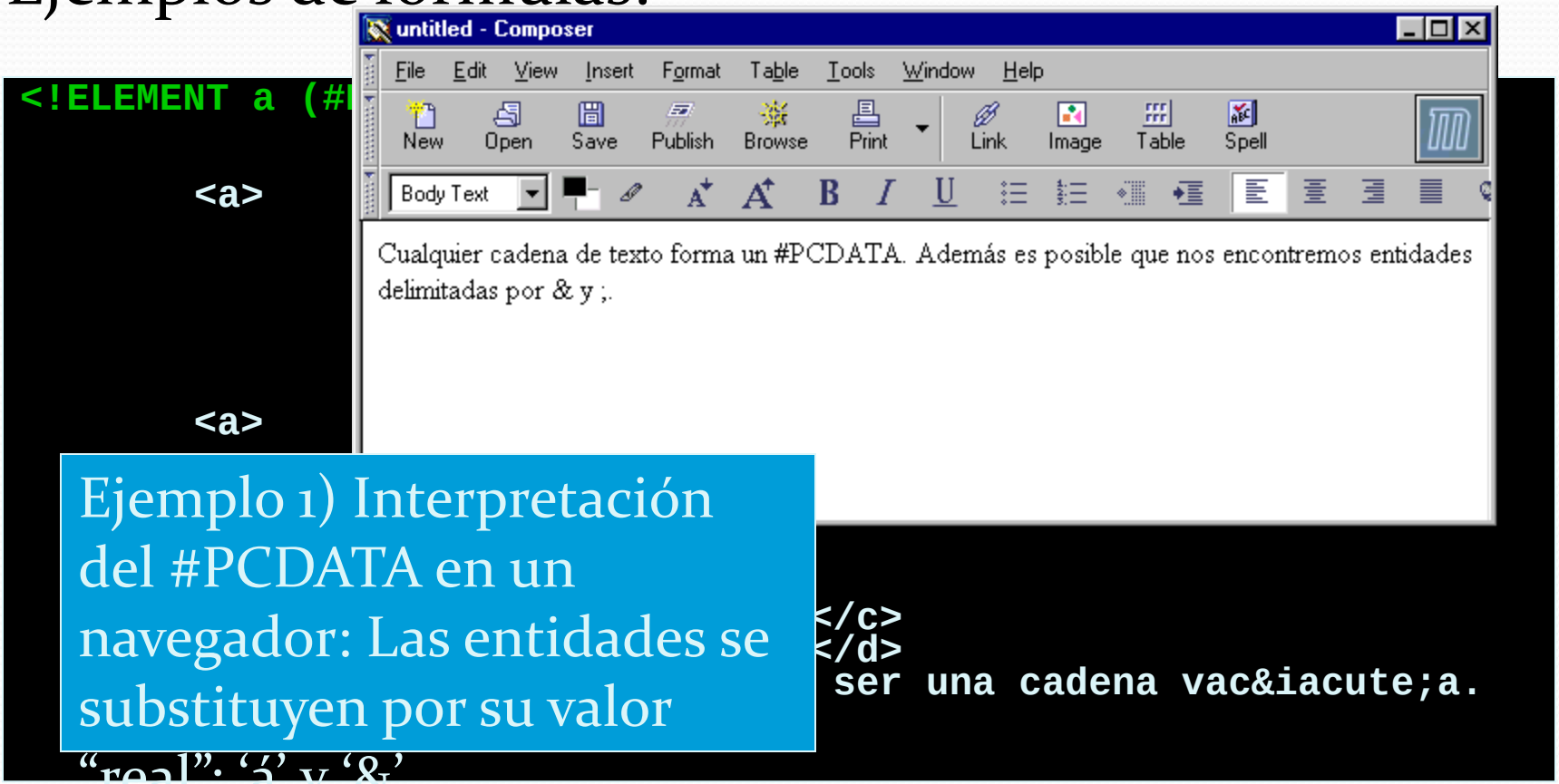
Elementos en DTD XML

Ejemplos de fórmulas:

```
<!ELEMENT a (#PCDATA, (b|c), d+, #PCDATA)>


        <a>
                Cualquier cadena de texto forma un #PCDATA.
                Adem&aacute;s es posible que nos encontremos
                entidades delimitadas por &amp; y ;.
                <b> contenido de b </b>
                <d> contenidio de d </d>
                Cualquier otra cadena de datos.
        <a>



        <a>
                <c> contenido de c </c>
                <d> contenido de d </d>
                Nota: #PCDATA puede ser una cadena vac&iacute;a.
        </a>
```

Elementos en DTD XML

Ejemplos de fórmulas:



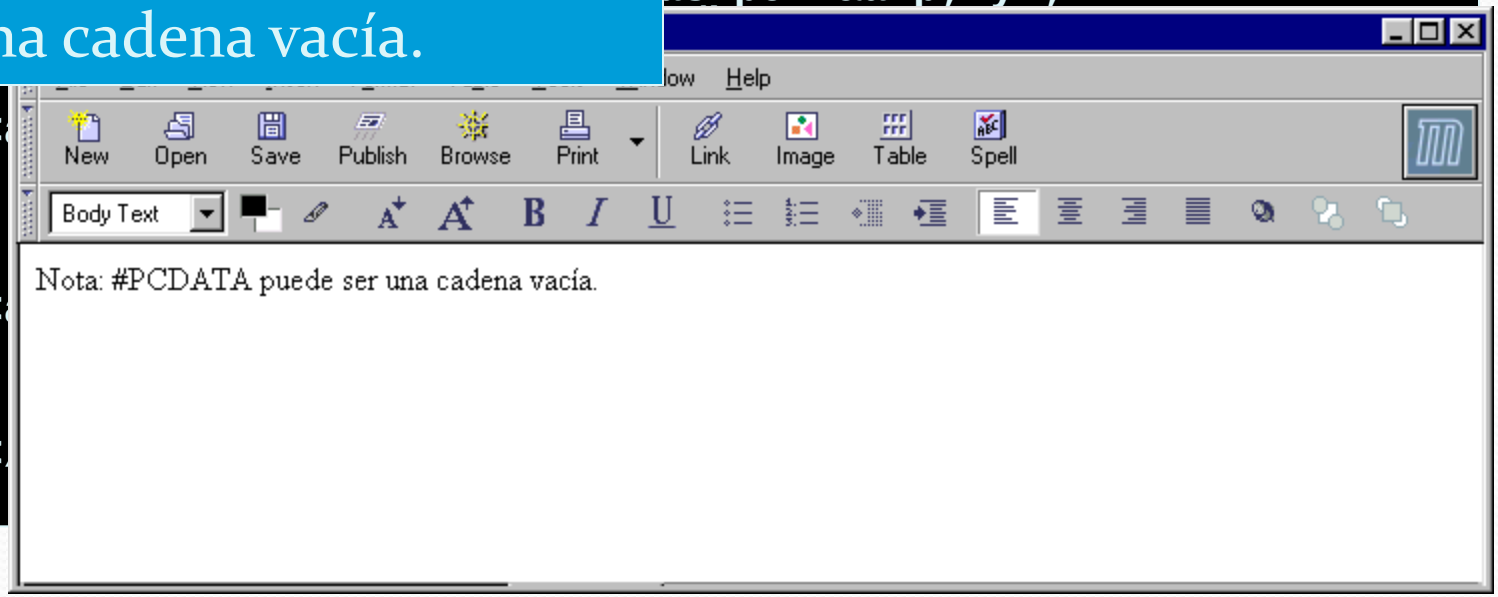
Ejemplo 1) Interpretación del #PCDATA en un navegador: Las entidades se substituyen por su valor "real": 'á' y '&'.

Elementos en DTD XML

Ejemplos de fórmulas:

```
<!ELEMENT a (#PCDATA, (b|c), d+, #PCDATA)>

        <a>
                Cualquier cadena de texto forma un #PCDATA.
                sible que nos encontremos
                as por &amp;amp; y ;.
```

Ejemplo 2) #PCDATA puede ser una cadena vacía.



Elementos en DTD XML

Ejemplos de fórmulas:

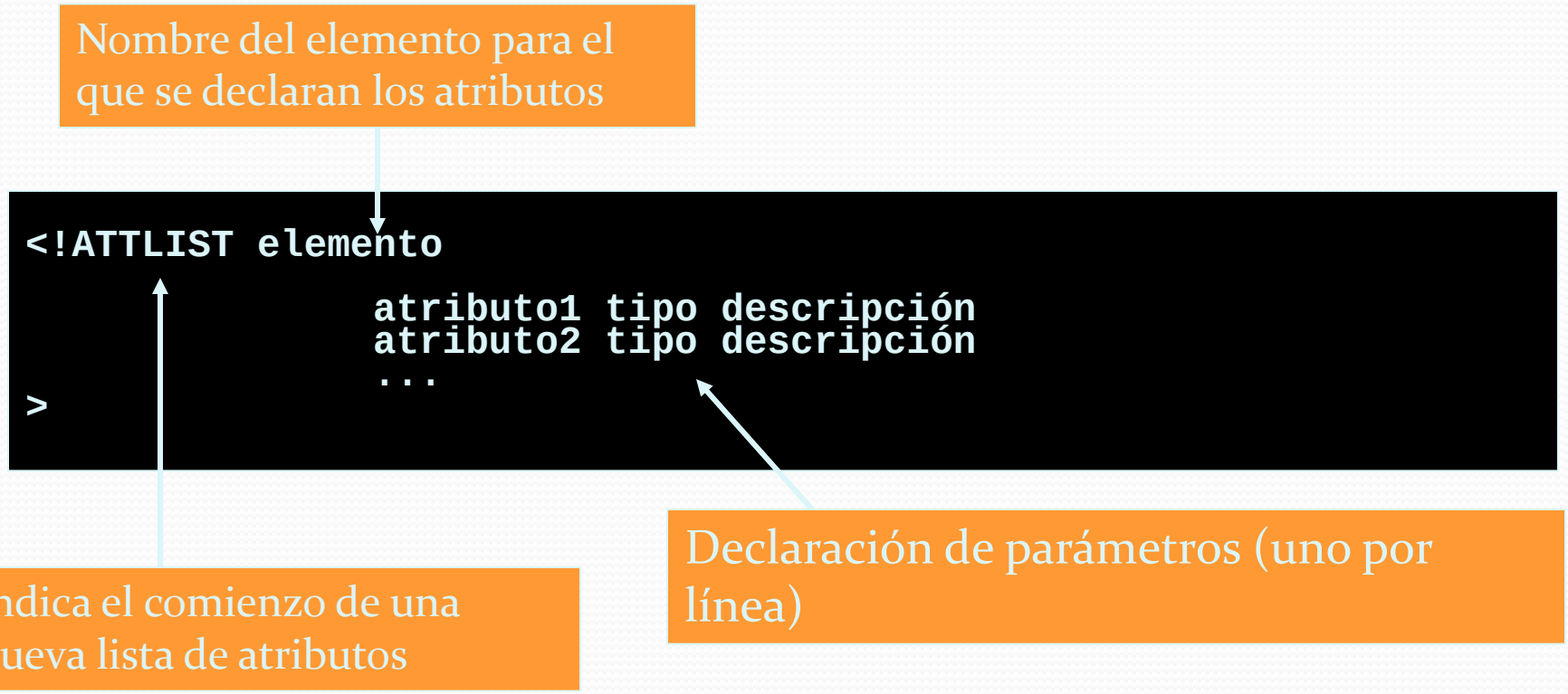
```
<!ELEMENT a (#PCDATA, (b|c), d+, #PCDATA)>

        <a>
                Cualquier cadena de texto forma un #PCDATA.
                Adem&aacute;s es posible que nos encontremos
                entidades delimitadas por &amp; y ;.
                <b> contenido de b </b>
                <d> contenidio de d </d>
                Cualquier otra cadena de
        <a>


        <a>
                <c> contenido de c </c>
                <d> contenido de d </d>
                Nota: #PCDATA puede ser
        </a>
```

VÁLIDA!

VÁLIDA!

Atributos en DTD XML

- Declaración de los atributos (parámetros):



Atributos en DTD XML

Tipos de parámetros:

- Indican los posibles valores que puede contener el atributo:

CDATA *(character data:* cualquier carácter y ciertas entidades, NO SERÁ "PARSEADO")

Nota: De momento no nos fijaremos en la *descripción*

```
<!ATTLIST a
        parametro CDATA descripción>
>

<a parametro="Cualquier cadena con entidades como &amp;"/>
```

NMTOKEN *(named token:* una cadena de letras, números o ciertos símbolos: como el nombre de una variable en Java)

```
<!ATTLIST a
        parametro NMTOKEN descripción>
>

<a parametro="Algo_sin_espacios_quizas_con_123456"/>
```

Atributos en DTD XML

NMTOKENS *(named token list:* secuencia de NMTOKEN separados por espacios)

```
<!ATTLIST a
        parametro NMTOKENS descripción>
>

<a parametro="Token1 Token2 Token3 ..."/>
```

ID *(unique identifier:* identificador que debe ser diferente para cada elemento, misma sintaxis que NMTOKEN)

```
<!ATTLIST a
        parametro ID descripción>
>

<a parametro="1"/>
<a parametro="2"/>
```

VÁLIDO!

```
<a parametro="uno"/>
<a parametro="uno"/>
```

NO VÁLIDO!

Atributos en DTD XML

IDREF *(identifier reference:* identificador único de otro elemento que debe existir en el documento)

```
<!ATTLIST a
        parametro IDREF descripción>
>

<!ATTLIST b
        otroparametro ID descripción>
>

<b otroparametro="uno"/>
<a parametro="uno"/>
```

VÁLIDO!

```
<b otroparametro="uno"/>
<a parametro="otro"/>
```

NO VÁLIDO!

Atributos en DTD XML

IDREFS *(identifier reference list:* lista de IDREF separados por espacios)

```
<!ATTLIST a
        parametro IDREFS descripción>
>

<!ATTLIST b
        otroparametro ID descripción>
>

<!ATTLIST c
        otroparametromas ID descripción>
>


<b otroparametro="ID1"/>
<b otroparametro="ID2"/>
<c otroparametro="ID3"/>

<a parametro="ID1 ID2 ID3 ..."/>
```

Atributos en DTD XML

ENTITY *(entity name:* nombre de una entidad general declarada en el documento)

```
<!ENTITY unaentidad SYSTEM "foto.png">

<!ATTLIST a
        parametro ENTITY descripción>
>

<a parametro="unaentidad"/>
```

Lo entenderemos mejor al estudiar entidades!

ENTITIES *(identifier reference list:* lista de ENTITY separadas por espacios)

```
<!ENTITY unaentidad SYSTEM "foto.png">
<!ENTITY otraentidad SYSTEM "foto2.png">

<!ATTLIST a
        parametro ENTITIES descripción>
>

<a parametro="unaentidad otraentidad"/>
```

Atributos en DTD XML

ENUMERACIÓN: Declaración especial (no usa palabra reservada):

(valor1 | valor2 | ...)

```
<!ATTLIST a
        parametro (valor1|valor2|...) descripción>
>

<a parametro="valor1"/>
<a parametro="valor2"/>

<a parametro="otrovalornoenumerado"/>
```

NO VÁLIDO!

Atributos en DTD XML

Descripción de parámetros:

- Pueden indicar valores por defecto.

```
<!ATTLIST a
        color NMTOKEN azul>
>

<a/>

<a color="rojo"/>


<!ATTLIST a
        color (azul|rojo) azul>
>

<a color="verde"/>
```

Atributos en DTD XML

Descripción de parámetros:

- Pueden indicar valores por defecto.

```
<!ATTLIST a
        color NMTOKEN azul>
>

<a/>

<a color="rojo"/>


<!ATTLIST a
        color (azul|rojo) azul>
>

<a color="verde"/>
```

No se indica nada del color, por tanto es azul

Se cambia el color a rojo

Definición no válida!

Atributos en DTD XML

Descripción de parámetros:

- Indican características especiales:

  #REQUIRED (el usuario debe proporcionar un valor siempre)

  #IMPLIED (el parámetro es opcional)

  #FIXED (debe aparecer y el usuario no puede alterarlo)

```
<!ATTLIST a
        identidad ID #REQUIRED>
        color (azul|rojo|verde) #IMPLIED>
        tipo (antiguo|nuevo) #FIXED nuevo
>

<a identidad="1"/>
<a identidad="2" color="verde"/>


<a color="rojo"/>
<a tipo="antiguo“/>
```

Entidades en DTD XML

- Son declaraciones abstractas que pueden usarse para definir otras partes de un documento XML (elementos, atributos, etc).
- Una definición menos formal: Son declaraciones especiales que se cambian por trozos de código.
  - Analogía con macros en lenguaje C:

```
#define CAPACIDAD 200

int a=CAPACIDAD;
```

```
<!entity CAPACIDAD "200">

<p>
La capacidad del sistema es
de &CAPACIDAD; bits por
segundo.
```

Entidades en DTD XML

- Según la aplicación, el formato del código y su ubicación existen varios tipos de entidades, cada una se declara con una sintaxis propia.

Entidades en DTD XML

- Entidades Genéricas:

```
<!ENTITY nombre #PCDATA>
```

  - Permiten la substitución de una cadena de texto. Ejemplo:

```
<!ENTITY minombre "Pepe Perez">

<p> Oí gritar mi nombre: &minombre; </p>
```

Entidades en DTD XML

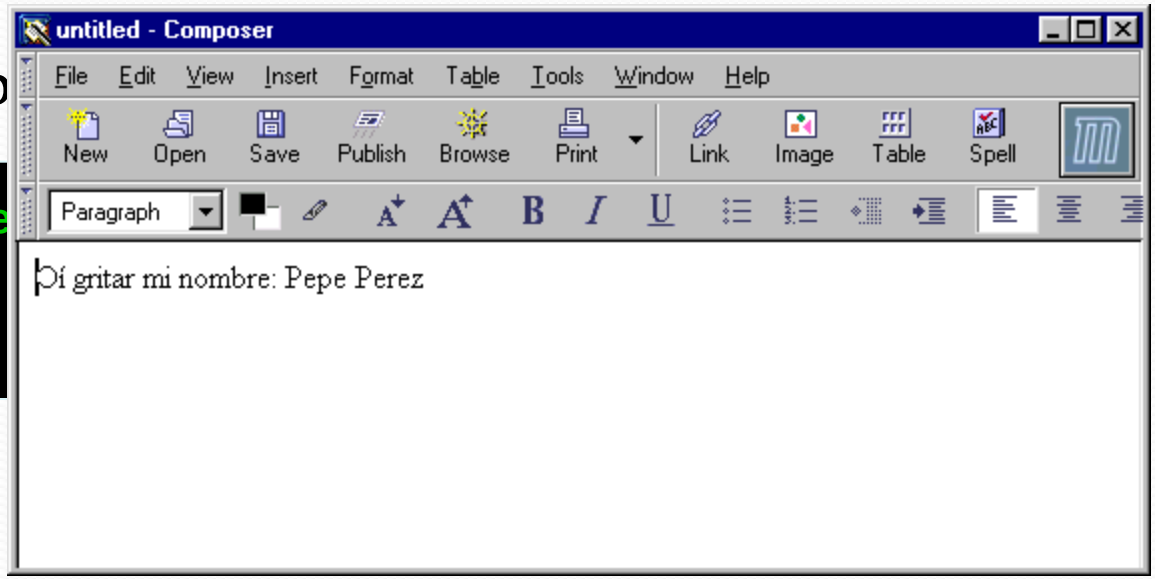
- Entidades Genéricas:

```
<!ENTITY nombre #PCDATA>
```

- Permiten la sub

Ejemplo:

```
<!ENTITY minombre

<p> Oí gritar mi
```



Entidades en DTD XML

- Entidades externas genéricas:

```
<!ENTITY nombre (PUBLIC|SYSTEM) URL>
```

  - Permiten la substitución por un texto contenido en una fuente externa. Ejemplo:

```
<!ENTITY minombre SYSTEM "/home/pepe/minombre.txt">

<p> Oí gritar mi nombre: &minombre; </p>
```

Entidades en DTD XML

- Entidades parámetro:
  - Las entidades estudiadas hasta ahora ejercían cambios en el **contenido** del XML.
  - Las entidades parámetro hacen lo mismo PERO EN EL PROPIO DTD → Ver ejemplos a continuación.

Entidades en DTD XML

- Entidades parámetro:

```
<!ENTITY % nombre (CDATA)>
```

- Entidades parámetro externas:

```
<!ENTITY % nombre (PUBLIC|SYSTEM) URL>
```

- Ahora se referencian como:

```
%nombre;    <--NO COMO &nombre;-->
```

Entidades en DTD XML

- Ejemplo:

```
<!ENTITY % contenidoppal "nombre|apellidos|direccion">
<!ENTITY % idattr "id ID #REQUIRED">

<!ELEMENT dni (%contenidoppal, datosnacimiento)>
<!ATTLIST dni
        %idattr;
        tipodni   (antiguo|nuevo|electonico) #REQUIRED nuevo
>
```

- Esto es como escribir:

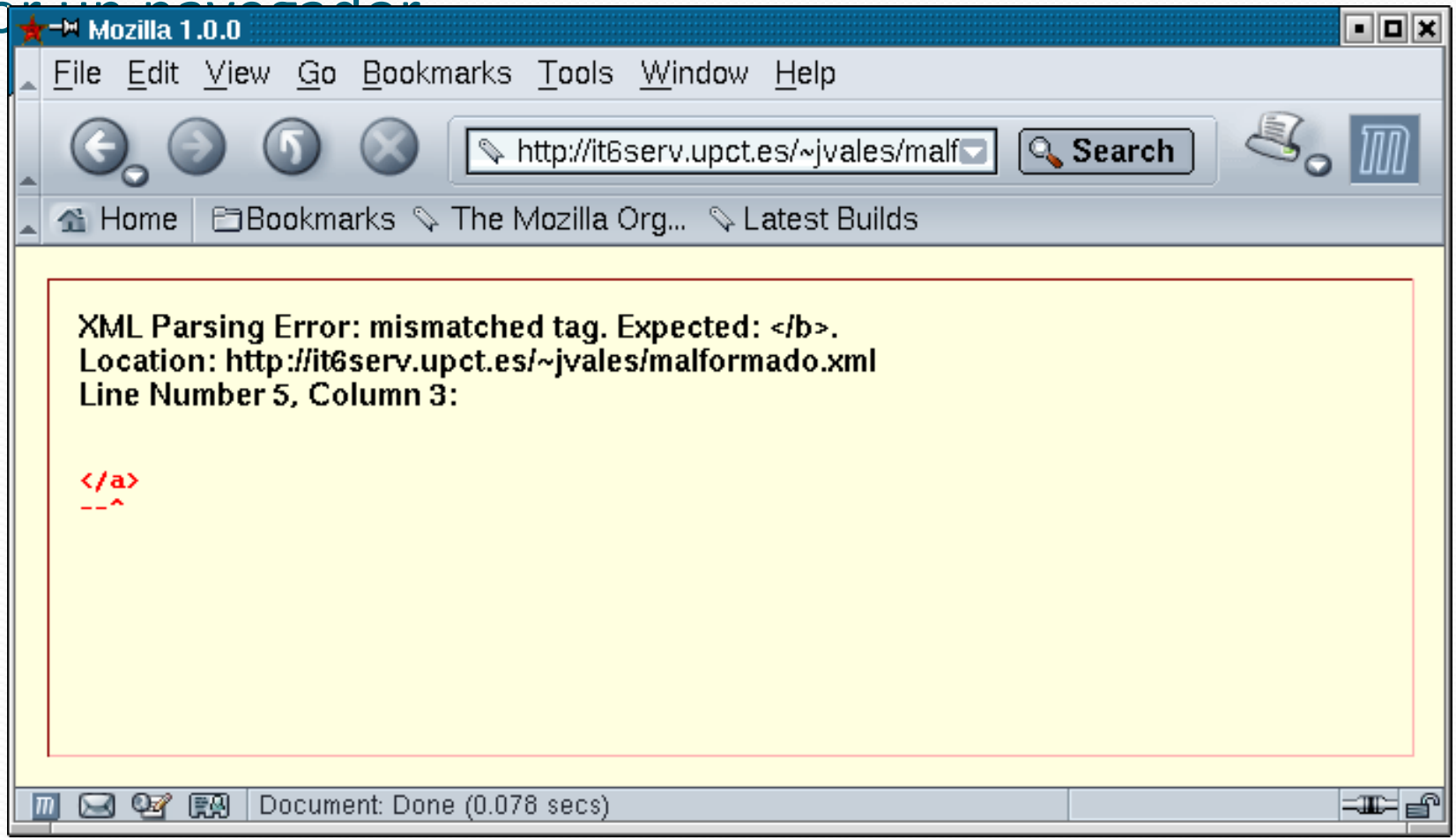
```
<!ELEMENT dni (nombre|apellidos|direccion, datosnacimiento)>
<!ATTLIST dni
        id      ID      #REQUIRED
        tipodni   (antiguo|nuevo|electonico) #REQUIRED nuevo
>
```

Interpretación de un documento XML por un navegador

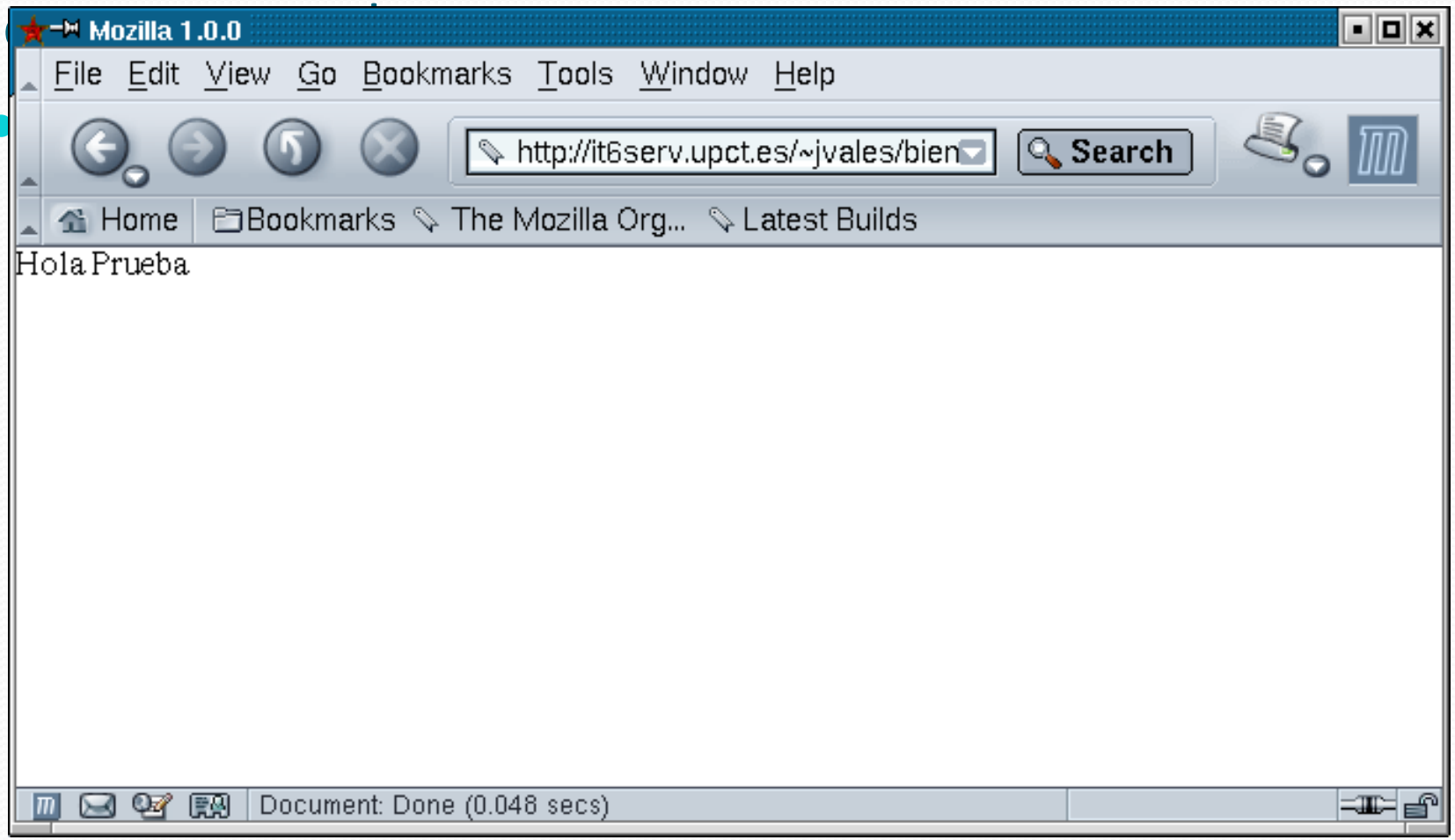
- El comportamiento final depende del navegador particular:
  - Al menos debe informar si el documento está bien formado o no. Ejemplo:

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

<a> Hola
<b> Prueba
</a>
</b>
```

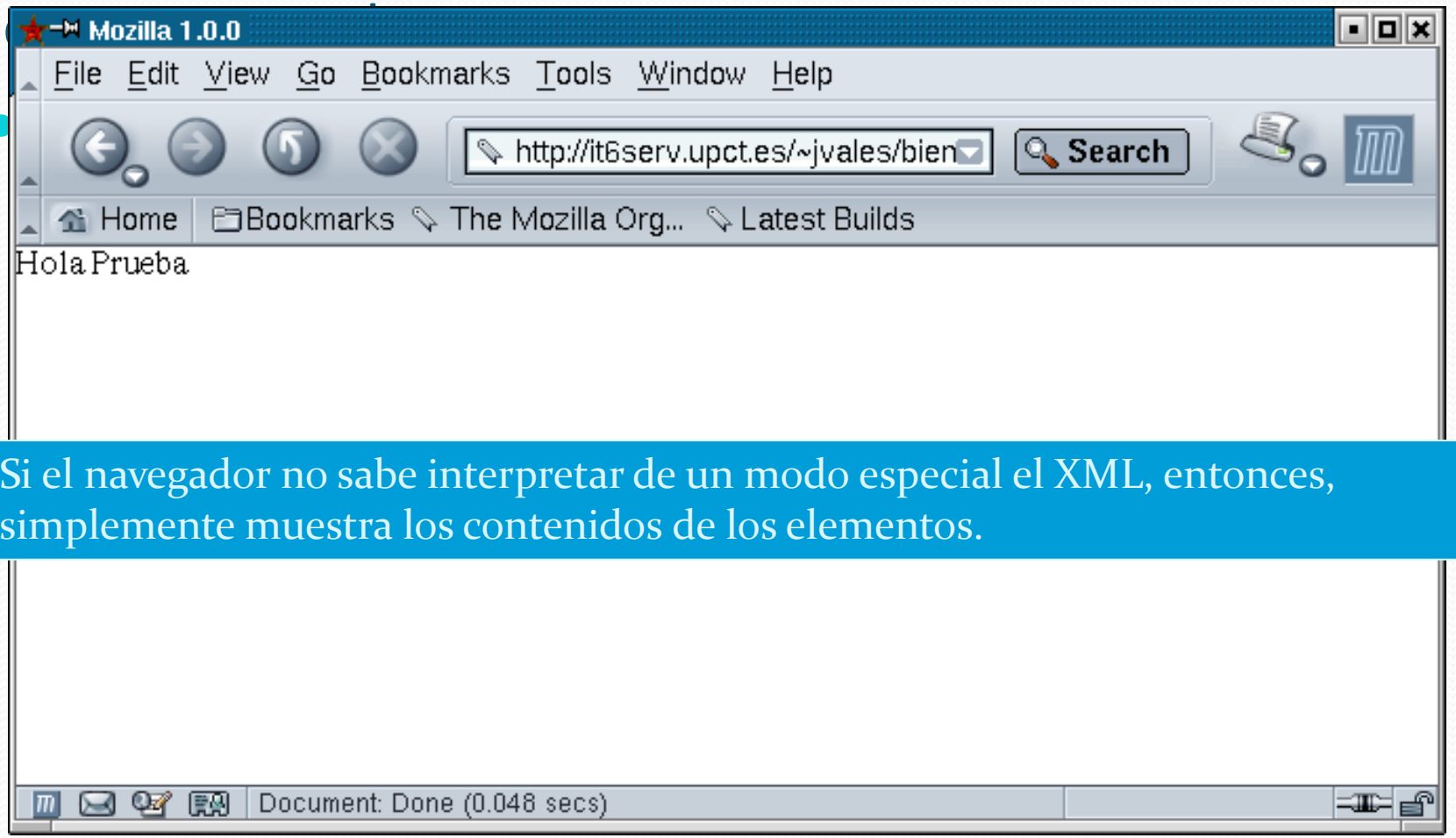
Interpretación de un documento XML por un navegador

- 

Interpretación de un documento XML p



Interpretación de un documento XML p



Si el navegador no sabe interpretar de un modo especial el XML, entonces, simplemente muestra los contenidos de los elementos.

Interpretación de un documento XML por un navegador

- Si un documento está especificado por su DTD, el navegador debería comprobar e informar si el XML es o no válido.

XHTML

- HTML es muy poco estricto y coherente en cuanto a la definición de las marcas: etiquetas de cierre opcionales, mayúsculas o minúsculas, comillas en los atributos, etc.
- XHTML es HTML conforme a las reglas sintácticas de XML
  - Documento debe estar bien formado
- XHTML hace que los documento sean más fáciles de procesar por una máquina

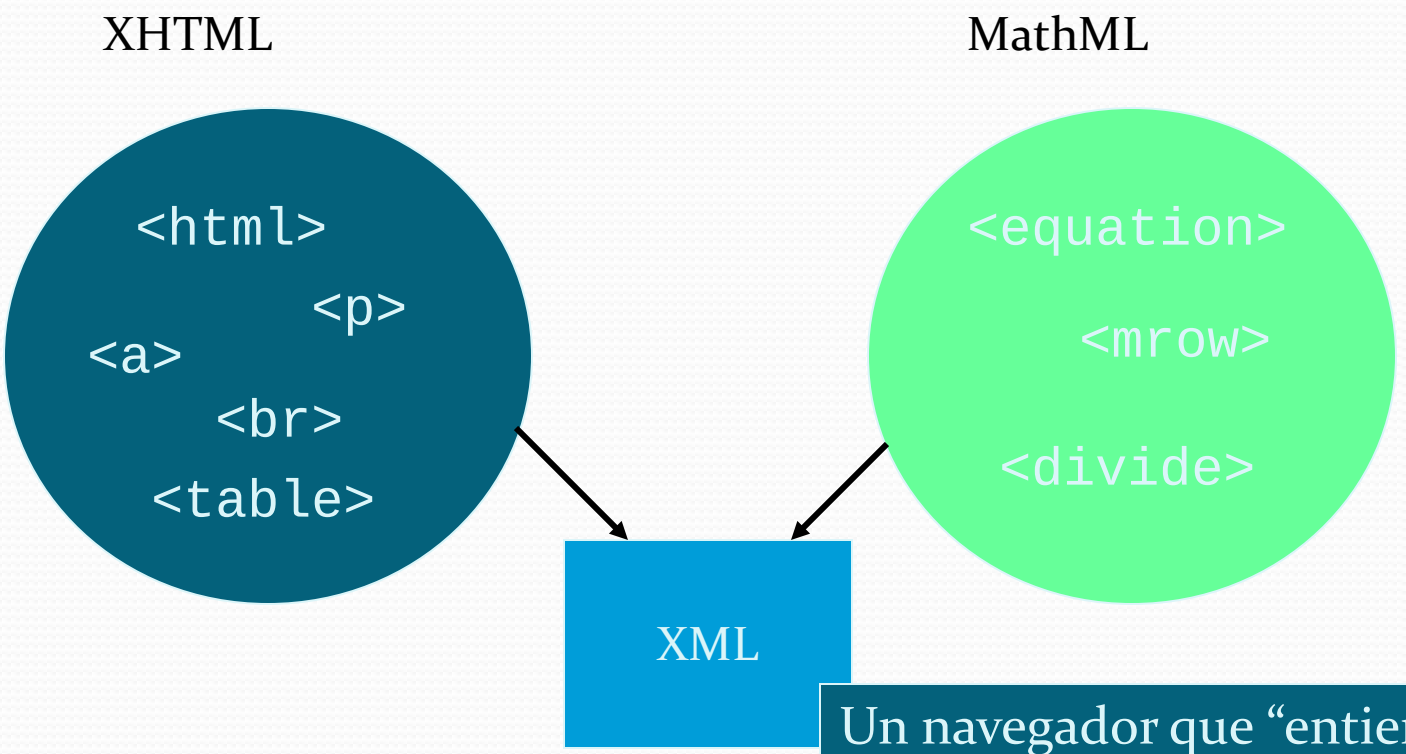
Tecnologías relacionadas con XML

Espacios de nombres

- El mecanismo de los espacios de nombres (*namespaces*) permite combinar en un mismo documento XML definiciones procedentes de varios tipos de documentos.
- Un *namespace* es un grupo de elementos y nombres de atributos. Cada grupo (*namespace*) se puede especificar y validar con su DTD correspondiente.

Espacios de nombres

- Ejemplo:



Espacios de nombres

- H

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

<html>

<p> Este documento combina dos lenguajes el XHTML y
el MathML.

Esto permite cosas <em> muy interesantes </em>.
Por ejemplo, escribir la ecuacion:
</p>

<math>
        <mrow>
        <mi> a </mi>
        <mo> + </mo>
        <mi> b </mi>
        <mo> = </mo>
        <mi> c </mi>
        </mrow>
</math>

</html>
```

Un navegador que “entienda” ambos lenguajes, mostrará correctamente este XML.

Espacios de nombres

- H

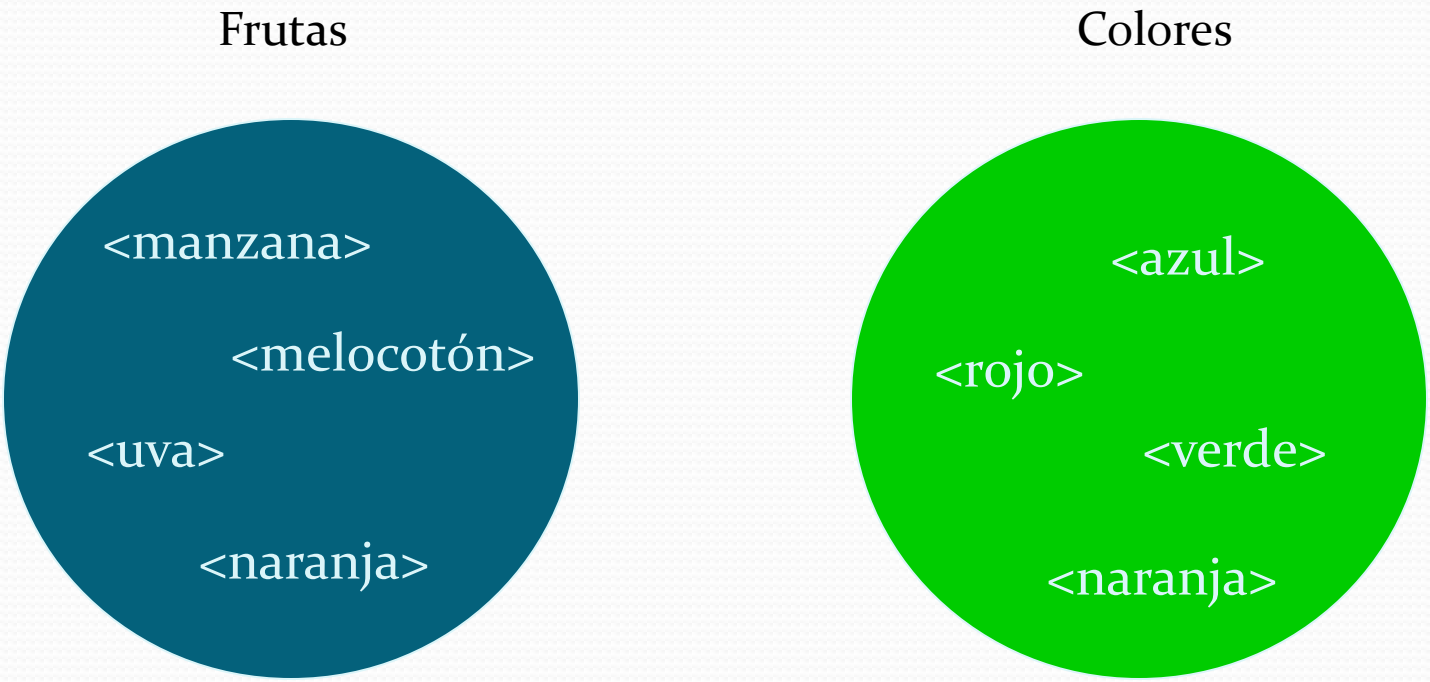
```
<?xml version="1.0"?>

<html>

<p> Este documento combina dos lenguajes el XHTML y
el MathML.

  <matrix>
    <matrixrow>
      <ci>x</ci>
      <ci>y</ci>
    </matrixrow>
    <matrixrow>
      <ci>z</ci>
      <ci>w</ci>
    </matrixrow>
  </matrix>
</reln>


</html>
```

¿ESTO ES CORRECTO?

NO!, ¿QUÉ SUCEDE SI DOS ELEMENTOS TIENEN EL MISMO NOMBRE EN DIFERENTES *NAMESPACES*?

nda"
rá

Espacios de nombres



Espacios de nombres

Frutas

Coloresu

```
<?xml version="1.0"?>
<manzana/>
...
<verde/>
...
<uva/>

<naranja/>
```

¿Es un color o una fruta?

Espacios de nombres

- Para evitar ambigüedades los tag se escriben:

```
prefijo_del_namespace:nombre_tag
```

El prefijo indica al interprete cuando conmutar de un lenguaje a otro.

Espacios de nombres

Frutas

Coloresu

```
<?xml version="1">
<fruta:manzana/>
...
<color:verde/>
...
<fruta:uva/>

<color:naranja/>
```

Un color!

Espacios de nombres

- Un namespace debe declararse antes de su uso:

```
<elemento xmlns:prefijo="URL">
```

Atributo válido para cualquier elemento. El *namespace* lo conoceran él y sus descendientes. Lo habitual es declarar los *namespace* conocidos en la `<raíz>`.

Espacios de nombres

- Un namespace debe declararse antes de su uso:

```
<elemento xmlns:prefijo="URL">
```

Prefijo con el que se identificaran los miembros del *namespace*. Puede ser cualquier palabra.

Espacios de nombres

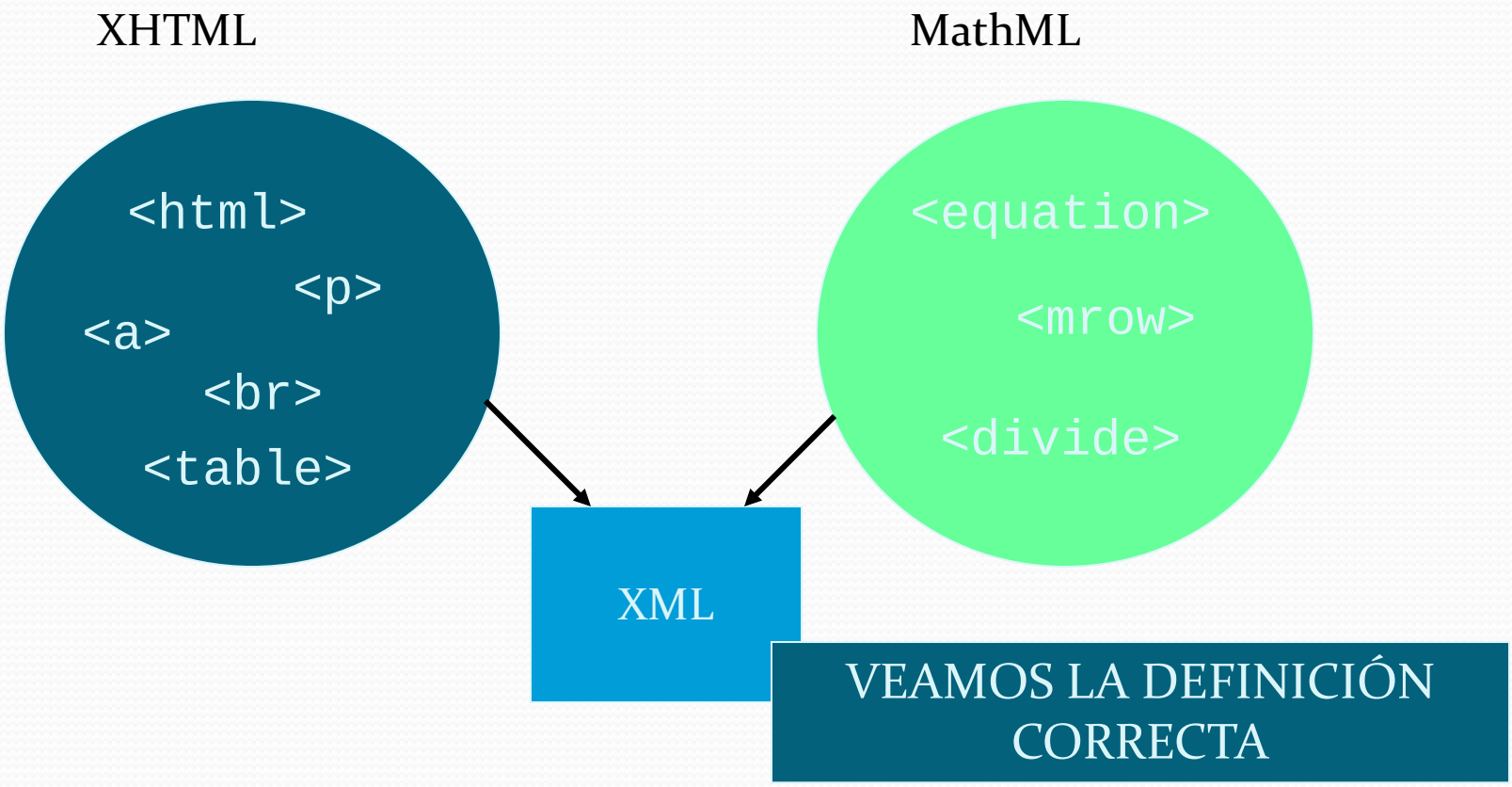
- Un namespace debe declararse antes de su uso: se declara en el elemento raíz

```
<elemento xmlns:prefijo="URL">
```

URL con la definición del *namespace:* su contenido será un DTD o similar.

Espacios de nombres

- Ejemplo:

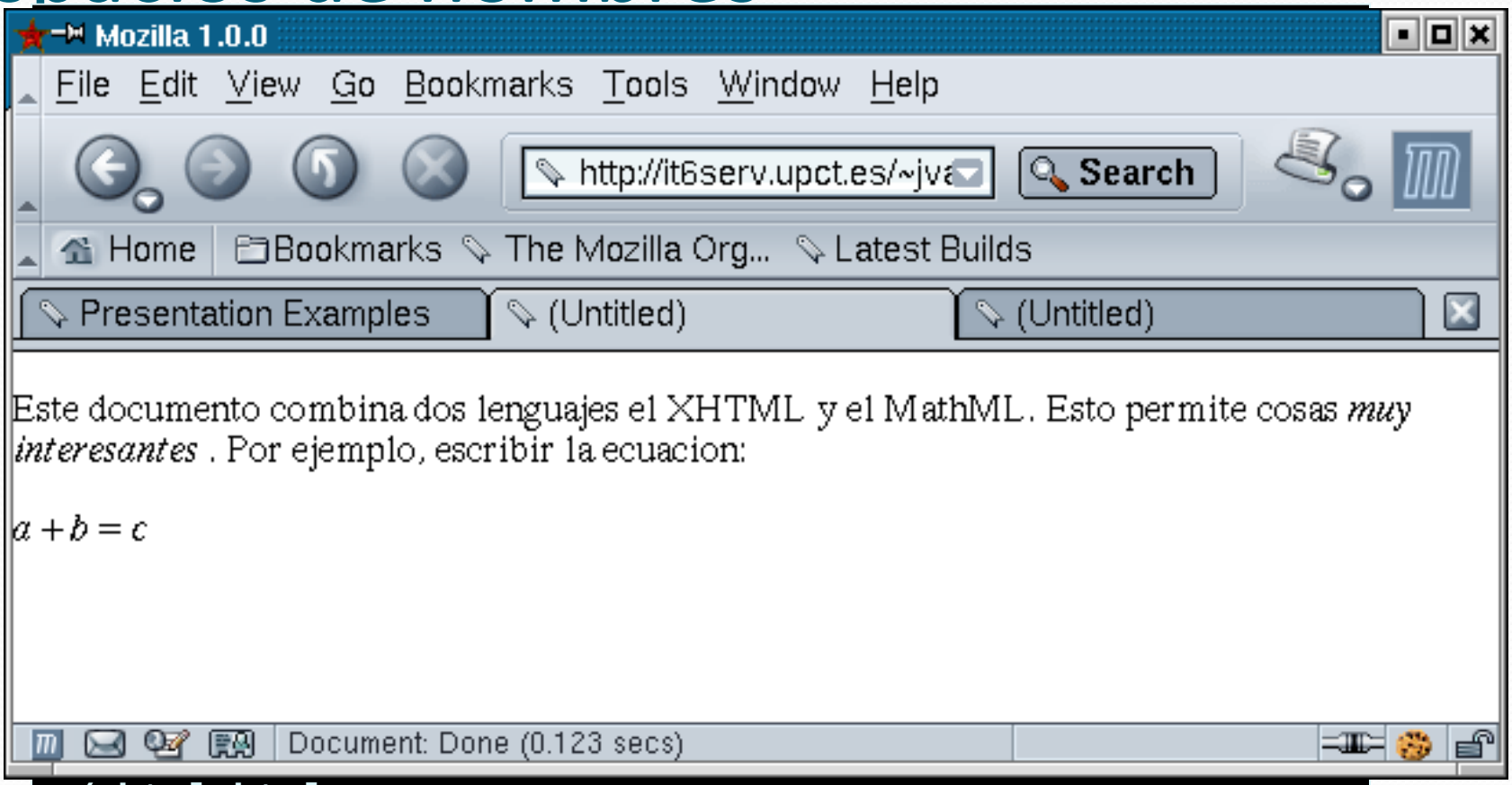


Espacios de nombres

- H

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

<xhtml:html
        xmlns:xhtml="http://www.w3.org/1999/xhtml"
        xmlns:math="http://www.w3.org/1998/Math/MathML“
>

<xhtml:p> Este documento combina dos lenguajes el XHTML
y el MathML.

Esto permite cosas <xhtml:em> muy interesantes
</xhtml:em>. Por ejemplo, escribir la ecuacion:
</xhtml:p>


<math:math>
        <math:mrow>
                <math:mi> a </math:mi>
                <math:mo> + </math:mo>
                <math:mi> b </math:mi>
                <math:mo> = </math:mo>
                <math:mi> c </math:mi>
        </math:mrow>
</math:math>

</xhtml:html>
```

Espacios de nombres

- 


```
</xhtml:html>
```

Espacios de nombres

- Pueden usarse también espacios de nombre "por defecto":

```
<elemento xmlns="URL">
```

Sin prefijo

Espacios de nombres

- Ejemplo:

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

<xhtml:html xmlns="http://www.w3.org/1999/xhtml">

<p> Este documento combina dos lenguajes el XHTML y el
MathML.

Esto permite cosas <em> muy interesantes </em>.
Por ejemplo, escribir la ecuacion:
</p>

<math:math xmlns:math="http://www.w3.org/1998/Math/MathML">
                <math:mi> a </math:mi>
                <math:mo> + </math:mo>
                <math:mi> b </math:mi>
                <math:mo> = </math:mo>
                <math:mi> c </math:mi>
        </math:mrow>
</math:math>

</html>
```

SINTAXIS MÁS COMPACTA.

XSCHEMA

- XML Schema
- Construcción alternativa a los DTD para especificar el formato de un documento XML.
- Un XSCHEMA es en si mismo un documento XML: ¡usamos XML para definir XML!

XSCHEMA

- Ventajas sobre DTD:
  - Más legible.
  - Permite **especificar el tipo de datos**
  - Permite especificar lenguajes XML que usen *namespaces* (con DTD es muy complejo).
  - Más posibilidades:
    - A nivel de elementos, por ejemplo: con XSCHEMA podemos indicar que un elemento aparece N veces.
    - A nivel de atributos, por ejemplo: puede especificarse que un atributo es un número, un dígito binario, etc.

XSCHEMA

- Ejemplo. Dado un XML (notas.xml) para definir notas:

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

<nota>

<para>Tove</para>
<de>Jani</de>
<asunto>Reminder</asunto>
<mensaje>Don't forget me this weekend!</mensaje>

</nota>
```

XSCHEMA

- Podemos escribir su DTD (notas.dtd) como:

```
<!ELEMENT nota (para, de, asunto, mensaje)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
```

XSCHEMA

- La definición alternativa con XSCHEMA sería (notas.xsd):

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

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

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

</xs:schema>
```

Lenguajes de estilo y transformación para XML

- XSL:
  - Lenguaje para crear hojas de estilo para documentos XML.
  - Más complejo que CSS: unos 60 selectores y 250 propiedades!!
- XSLT:
  - Lenguaje para crear transformaciones en los documentos.
    - XML → HTML
      → WML
      → etc

Cada transformación usa una hoja XSLT diferente.

Lenguajes de estilo y transformación para XML

- XPATH:
  - Lenguaje que permite identificar partes de un documento (concepto similar a `#enlace_interno` en URL).
- XPOINTER:
  - Lenguaje para definir punteros a puntos o partes de un documento (concepto similar a URL).
- XLINK
  - Mecanismo de hiperenlaces (mas general que las <a> de HTML).
  - Puede conectar varios objetos en ambas direcciones.

Referencias y bibliografía

- Libros:
  - "Learning XML", *Erik T. Ray,* O'Reilly, 2001.
    Disponible en:
    http://labit501.upct.es/ad/libros

- Guía rápida:
  - "XML Syntax Quick Reference", *Mulberry Technologies, Inc.,* 2000.

Referencias y bibliografía

- WWW:
  - http://www.w3.org/XML/ → Página principal de desarrollo de la tecnología XML
  - http://www.w3.org/TR/2004/REC-xml11-20040204/ → Recomendación de XML 1.1
  - http://www.w3.org/TR/REC-xml-names/ → Espacios de nombres
  - http://www.w3.org/TR/xsl/ → Hojas de estilo en XML

Referencias y bibliografía

- WWW:
  - http://www.w3.org/XML/ → Página principal de desarrollo de la tecnología XML
  - http://www.w3.org/TR/2004/REC-xml11-20040204/ → Recomendación de XML 1.1
  - http://www.w3.org/TR/REC-xml-names/ → Espacios de nombres
  - http://www.w3.org/TR/xsl/ → Hojas de estilo en XML

Referencias y bibliografía

- Tutoriales:
  - http://www.w3schools.com/xml/ → Varios tutoriales tecnologías XML
  - http://www.dat.etsit.upm.es/~abarbero/curso/xml/xmltutorial.html → Tutorial en español, con bastantes enlaces