

# **Transmissió d'informació per mitjà del web. Mecanismes de validació de documents XML**

Xavier Sala

**Llenguatges de marques i sistemes de gestió  
d'informació**



# Índex

<b>Introducció</b>	<b>5</b>
<b>Resultats d'aprenentatge</b>	<b>7</b>
<b>1 Utilització dels llenguatges de marques en entorns web</b>	<b>9</b>
1.1 Cascading style sheets (CSS) . . . . .	10
1.1.1 Navegadors web . . . . .	11
1.1.2 Associar un full d'estil a un document . . . . .	13
1.1.3 Regles CSS . . . . .	16
1.1.4 Model de caixes . . . . .	27
1.1.5 Propietats . . . . .	34
1.1.6 Amagar contingut . . . . .	40
1.1.7 Afegir contingut a l'XML . . . . .	42
1.1.8 Validació . . . . .	47
1.2 HTML / XHTML . . . . .	48
1.2.1 HTML . . . . .	49
1.2.2 Convertir d'HTML a XHTML . . . . .	58
1.2.3 Eines de disseny web . . . . .	60
<b>2 Definició d'esquemes i vocabularis en XML</b>	<b>65</b>
2.1 Validació de documents XML . . . . .	65
2.2 Processadors d'XML . . . . .	67
2.2.1 Biblioteques per validar XML . . . . .	67
2.2.2 Programes . . . . .	70
2.3 DTD . . . . .	73
2.3.1 Associar una DTD a un document XML . . . . .	73
2.3.2 Definició d'esquemes amb DTD . . . . .	78
2.3.3 Limitacions . . . . .	90
2.3.4 Exemple de creació d'una DTD . . . . .	92
2.4 W3C XML Schema Definition Language . . . . .	95
2.4.1 Associar un esquema a un fitxer XML . . . . .	95
2.4.2 Definir un fitxer d'esquema . . . . .	96
2.4.3 Exemple de creació d'un XSD . . . . .	108
2.5 Conversió entre esquemes . . . . .	113
2.5.1 Trang . . . . .	113



## Introducció

Els llenguatges basats en XML estan pensats per ser usats per màquines però sense oblidar que també se n'ha de facilitar la lectura per als humans. En aquesta unitat es veuran tecnologies destinades a facilitar la visualització de les dades en fitxers basats en XML i a facilitar-ne l'ús per part dels programes.

Els documents XML no estan pensats per ser visualitzats pels humans però algunes tecnologies en faciliten la visualització. L'apartat “Utilització dels llenguatges de marques en entorns web” consistirà a mostrar sistemes per a la representació de documents en formats més “amigables” per als humans, com són CSS i XHTML.

En l'apartat “Definició d'esquemes i vocabularis en XML” veureu que a pesar de la llibertat que donen els llenguatges de marques a l'hora de crear elements perquè puguin ser usats per programes caldrà restringir-los d'alguna manera. Veureu els dos vocabularis més importants destinats a restringir quines són les etiquetes que s'hi poden fer servir: DTD i XML Schemas.



## Resultats d'aprenentatge

En acabar aquesta unitat, l'alumne:

**1. Utilitza llenguatges de marques per a la transmissió d'informació per mitjà del Web analitzant l'estructura dels documents i identificant-ne els elements**

- Identifica i classifica els llenguatges de marques relacionats amb el Web i les seves versions diferents.
- Analitza l'estructura d'un document HTML i identifica les seccions que el componen.
- Reconeix la funcionalitat de les principals etiquetes i atributs del llenguatge HTML.
- Estableix les semblances i diferències entre els llenguatges HTML i XHTML.
- Reconeix la utilitat d'XHTML en els sistemes de gestió d'informació.
- Utilitza eines en la creació del web.
- Identifica els avantatges que aporta la utilització de fulls d'estil.
- Aplica fulls d'estil.

**2. Estableix mecanismes de validació per a documents XML utilitzant mètodes per definir-ne la sintaxi i estructura.**

- Estableix la necessitat de descriure la informació transmesa en els documents XML i les seves regles.
- Identifica les tecnologies relacionades amb la definició de documents XML.
- Analitza l'estructura i sintaxi específica utilitzada en la descripció.
- Crea descripcions de documents XML.
- Utilitza descripcions en l'elaboració i validació de documents XML.
- Associa les descripcions de documents XML amb els documents XML.
- Utilitza eines específiques de validació.
- Documenta les descripcions de documents XML.



## 1. Utilització dels llenguatges de marques en entorns web

A l'hora de crear documents XML només cal preocupar-se de les etiquetes, de l'estructura dels documents i de comprovar que el document compleixi les normes del vocabulari correcte, però no de com es veuran aquestes pàgines.

Tot això està bé perquè un dels objectius principals és fer que els programes puguin interpretar les dades que contenen els documents. Alhora, si s'han seguit les recomanacions qualsevol persona amb una mica d'esforç també podrà interpretar les dades que conté el document.

XML està pensat perquè un usuari pugui entendre les dades però no per ser llegides còmodament. Algú que no sigui expert en informàtica, en intentar veure les dades d'un document no espera veure això:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/css" href="notes.css" ?>
3  <institut nom="Institut Cendrassos">
4      <cicles>
5          <especialitat nom="Informàtica">
6              <cicle id="ASIX">
7                  Administració de Sistemes Informàtics i Xarxes
8              </cicle>
9              <cicle id="SMX" >
10                 Sistemes Microinformàtics i Xarxes
11             </cicle>
12         </especialitat>
13     </cicles>
14     <notes>
15         <classe nom="ASIX">
16             <alumne aprovat="NO"><nom>Jaume Capmany</nom></alumne>
17             <alumne aprovat="SI"><nom>Mohamed Polih</nom><nota>5</nota></alumne>
18                 >
19             <alumne aprovat="SI"><nom>Juan Pérez</nom><nota>6</nota></alumne>
20             <alumne aprovat="SI"><nom>Frederic Pi</nom><nota>5</nota></alumne>
21             ...

```

Per contra, el que aquest usuari espera trobar-se és més aviat una cosa com la de la figura 1.1.

**FIGURA 1.1.** Document XML amb un estil aplicat

Administració de Sistemes Informàtics i Xarxes		
<b>IOC</b>	institut obert de catalunya	
Jaume Capmany		
Mohamed Polih	5	
Juan Pérez	6	
Frederic Pi	5	
Bernat Alegria	9	
Samuel Barnadas		
Felip Martí	5	
Juanjo Roig		
Magdalena Serra	10	

Si bé és cert que hi ha vocabularis XML pensats per ser representats, com SVG (figura 1.2) o MathXML, aquests llenguatges no estan pensats per representar informació general sinó per representar camps concrets.

**FIGURA 1.2.** SVG està pensat per representar gràfics vectorials en 2D.



D'adaptar el contingut d'un document per fer que pugui ser visualitzat d'una manera alternativa s'encarregaran **els fulls d'estil**.

Els fulls d'estil seran els responsables de fer que els documents XML puguin ser visualitzats de maneres diferents per adaptar-se al medi per al qual seran visualitzats.

En documents XML els fulls d'estil més usats són:

- **Cascading style sheets (CSS)**: un mecanisme simple per afegir estil als documents. És el que es fa servir en HTML/XHTML.
- **XSL formatting objects (XSL-FO)**: un llenguatge XML que només té com a objectiu donar format als documents XML. Es fa servir sobretot per transformar la informació en altres formats com PDF (*portable document format*) o HTML.

## 1.1 Cascading style sheets (CSS)

XHTML és la versió en XML d'HTML.

Els fulls d'estil CSS van ser creats per posar una mica d'ordre en l'entorn web i, per tant, es van pensar per donar format a HTML (i per tant, també a XHTML).

De totes maneres també es poden aplicar a documents XML que no siguin XHTML, tot i que normalment portarà una mica més de feina. Com que les etiquetes HTML ja defineixen una manera de ser representades per defecte això permet que en alguns casos no calgui canviar aquesta representació, però en canvi en XML no hi ha informació sobre com s'ha de presentar la informació i, per tant, normalment s'haurà de definir la manera de representar cada una de les etiquetes.

Han sortit diverses versions de CSS:

- **CSS1:** va ser la primera versió que va sortir i ja no està suportada pel W3C.
- **CSS2:** va expandir les característiques de CSS1 afegint-li moltes funcionalitats. Tot i que va sortir el 1998 encara és la versió més suportada pels programes i està en constant revisió.
- **CSS3:** CSS3 a més d'afegir noves funcionalitats converteix l'especificació en modular, de manera que alguns mòduls s'han convertit en recomanacions i d'altres encara no. No està gaire suportada.

Mentre que fa temps tothom visitava les pàgines web amb un ordinador, en un moment donat la situació va començar a canviar radicalment i es van començar a visitar les pàgines web amb altres dispositius: ordinadors de butxaca, telèfons mòbils, televisions, etc... i les primeres versions d'HTML no estaven preparades per al canvi.

En l'especificació de la versió 2 de CSS fins i tot s'hi inclou un punt, que anomena "A brief CSS 2.1 tutorial for XML", en què es parla com es pot lligar CSS amb documents XML.

Calia que les pàgines s'adaptessin al dispositiu que les visitava, ja que allò que en un ordinador personal era perfectament llegible, en un telèfon mòbil podia ser il·legible. Però en HTML era molt difícil convertir una pàgina que tenia un format específic en un altre, ja que les dades estaven mesclades amb la manera de representar-les. **Calia separar les dades de la manera de representar-les.**

A més, la popularitat de les pàgines web va començar a fer que els llocs web cada vegada tinguessin més pàgines. En tenir el format dins del document, si es volia que totes les pàgines mostressin un aspecte coherent s'havia de repetir l'estil en totes les pàgines. **Calia una manera de poder aplicar un estil a múltiples pàgines.**

CSS intenta donar una manera de definir les regles de presentació d'un document sense barrejar les dades amb el contingut i permetent que els estils es puguin aplicar a múltiples pàgines.

Per tant, CSS proporciona un sistema per controlar la presentació de les pàgines sense haver de canviar el document al qual es vol donar estil.

### 1.1.1 Navegadors web

Com que CSS va sorgir de les tecnologies web els programes que en tenen un suport més gran són els **navegadors web**. Per tant, solen ser els programes més usats per visualitzar documents als quals s'aplica un full d'estil.

Actualment els navegadors web estan fent tasques molt diferents d'aquelles per a les quals van ser pensats inicialment. Els navegadors web no solament es fan servir per visualitzar informació en una pàgina web sinó que es poden fer servir per descarregar programes, visualitzar pel·lícules, escoltar música, edició de textos,

executar programes, etc. Fins i tot les tecnologies en el núvol ens permeten administrar servidors des del navegador.

- La complexitat associada a les múltiples tasques noves que han anat assumint els navegadors ha fet que no sempre compleixin perfectament tots els estàndards i recomanacions.
- A vegades algunes especificacions no deixen clar què s'ha de fer en casos determinats i els navegadors han tendit a improvisar, i això fa que no tots els navegadors responguin igual en aquestes situacions.

El suport dels navegadors a CSS ha estat molt variable al llarg del temps però en general els navegadors interpretaran les regles CSS majoritàriament de la mateixa manera (però no sempre) i, per tant, en crear un full d'estil CSS s'haurà de comprovar com es veu amb diferents navegadors per assegurar-se que no té problemes.

## Visualització de documents XML

En carregar un document XML que no tingui full d'estil els navegadors solen carregar un full d'estil per defecte que sol ser en forma d'arbre com en la figura 1.3.

**FIGURA 1.3.** Visualització per defecte d'un document XML en el Firefox

```

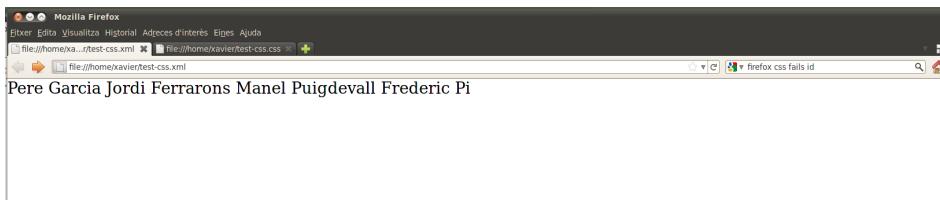
<!DOCTYPE html>
<html>
<head>
<title>Mozilla Firefox</title>
</head>
<body>
<p>Aquest fitxer XML no sembla que tingui cap informació d'estil associada. Es visualitza a sota l'esquema en arbre del document.</p>
<pre>
- <classe>
  - <alumnes>
    - <alumne aprovat="si" delegat="si">
      <nom>Pere </nom>
      <cognom>Garcia</cognom>
    </alumne>
    - <alumne aprovat="si">
      <nom>Jordi</nom>
      <cognom>Ferrarons</cognom>
    </alumne>
    - <alumne aprovat="no">
      <nom>Manel</nom>
      <cognom>Puigdevall</cognom>
    </alumne>
    - <alumne aprovat="si">
      <nom>Frederic</nom>
      <cognom>Pi</cognom>
    </alumne>
  </alumnes>
</classe>
</pre>
</body>
</html>

```

Aquesta visualització avisa que el document no té full d'estil.

Si es defineix un full d'estil sense regles es mostrerà el document amb la visió per defecte (figura 1.4).

La representació per defecte d'un document XML és simplement ignorar els elements i pintar el contingut de les dades un rere l'altre.

**FIGURA 1.4.** Visualització per defecte d'un document XML en el Firefox

### 1.1.2 Associar un full d'estil a un document

Per poder aplicar un full d'estil CSS a un document aquest s'ha de modificar per indicar-li quin és el full d'estil per aplicar. Associar un full d'estil es farà de manera diferent si l'estem associant a un document HTML que si l'estem associant a un document XML.

Com que l'XHTML és HTML i a més està basat en XML, es pot associar un full d'estil CSS a XHTML fent servir qualsevol dels sistemes.

#### Associar un document HTML

L'HTML defineix dues etiquetes per mitjà de les quals es poden definirfulls d'estils per a un document:

1. <style>
2. <link>

Totes dues regles s'han de definir dins de l'etiqueta <head> del document.

L'element <style> permet que es puguin afegir regles CSS directament en un document HTML.

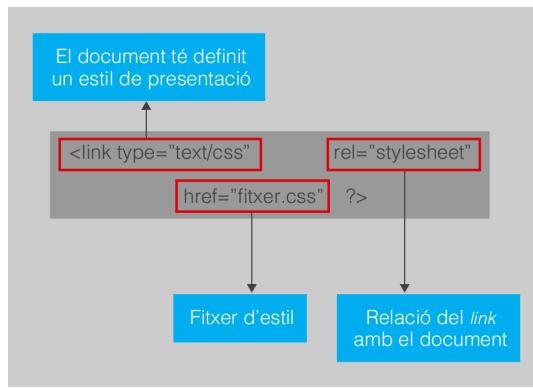
```

1  <html>
2    <head>
3      <title>Test</title>
4      <style type="text/css">
5          ... regles ...
6      </style>
7  ...

```

Les regles CSS simplement s'han d'especificar entre l'obertura i el tancament de l'element.

Aquest no és el sistema més usat, ja que es perd la possibilitat de fer que totes les pàgines d'un lloc web concret tinguin un estil semblant. Per aquest motiu el més corrent és definir el full d'estil com un fitxer a part amb l'element <link> (figura 1.5).

**FIGURA 1.5.** Definir un full d'estil CSS extern en HTML

### Els tipus MIME

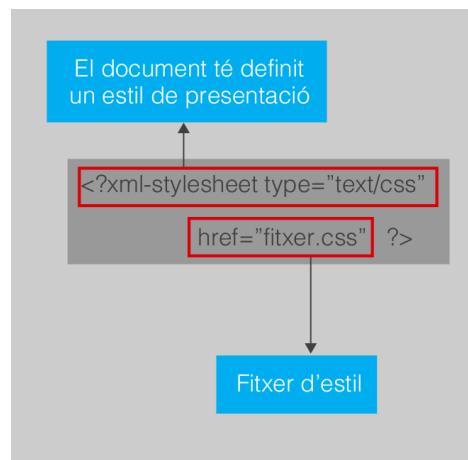
MIME és un estàndard que es fa servir per transferir arxius binaris en protocols que només accepten la transferència de caràcters de text. Inicialment va ser creat per poder adjuntar arxius per correu electrònic, però actualment hi ha molts altres protocols que el fan servir.

- L'atribut `type` sempre serà el valor MIME de CSS (`text/css`).
- `rel` serveix per determinar quina és la relació de l'arxiu que estem enllaçant amb el document.
- I l'atribut `href` és el que es fa servir per enllaçar amb l'arxiu que conté les regles CSS. Normalment se li posa extensió `.css` però pot tenir l'extensió que es vulgui.

### Associar un document XML

L'XML permet a l'usuari especificar les seves etiquetes pròpies i no en porta cap de predefinida, o sigui que en XML les etiquetes `<style>` i `<link>` no existiran, o si existeixen estarán dotades d'un significat diferent del que tenen en HTML.

Per aquest motiu els fulls d'estil en documents basats en XML es defineixen fent servir la instrucció de procés `xmlstylesheet` (figura 1.6).

**FIGURA 1.6.** Definició de la instrucció de procés "xmlstylesheet"

La majoria dels atributs tenen el mateix nom que en l'element `<link>` d'HTML (excepte `rel`, que aquí no es pot fer servir).

## Altres paràmetres

Tant <link> com <?xmlstylesheet ?> permeten tota una sèrie d'atributs a part dels dos bàsics (type i href) que tenen per objectiu passar més informació al programa per representar el document (taula 1.1).

**TAULA 1.1.** Paràmetres de càrrega de CSS

Atribut	Descripció
<code>title</code>	El nom o el títol del full.
<code>media</code>	Indica sota quin medi s'ha de carregar el CSS.
<code>charset</code>	Codi de caràcters que fa servir el full d'estil.
<code>alternate</code>	Indica si el CSS és preferit o no.

Sense cap mena de dubte, dels atributs que hi ha el més usat és `media`, que permet que s'apliqui un estil o un altre en funció del dispositiu amb què s'està intentant visualitzar el document.

Gràcies a l'atribut `media` es poden ferfulls d'estil que s'aplicaran en funció de quin sigui el dispositiu que vol llegir el document (taula 1.2).

**TAULA 1.2.** Valors possibles en l'atribut "media"

valor	dispositius
<code>screen</code>	Per ser visualitzat en un monitor de PC
<code>print</code>	Sortida en paper: impressores, etc.
<code>tty</code>	Sortida en terminals, teletips, etc.
<code>handheld</code>	PDA, telèfons mòbils, etc.
<code>braille</code>	La sortida es generarà en dispositius tàctils per a cecs.
<code>aural</code>	Per sortir en lectors de pantalla, sintetitzadors de veu, etc.
<code>tv</code>	Televisors, consoles de videojocs, etc.

Tot i que el suport en els navegadors està millorant, normalment (excepte per als atributs `screen` i `print`) els navegadors no sempre compleixen l'estàndard i no hi ha gaires garanties que si es navega per la pàgina amb un navegador de telèfon mòbil, per exemple, aquesta faci servir el full d'estil marcat com a `handheld` i no pas un altre.

Si no s'especifica cap valor a l'atribut `media` s'està indicant que el que s'està definint és el full d'estil per defecte.

```

1  <?xmlstylesheet href="web.css"?>
2  <?xmlstylesheet media="print" href="imprimir.css"?>
3  <?xmlstylesheet media="handheld" href="mobil.css"?>
```

En cas d'especificar diversosfulls d'estil es barrejaran les regles de tots i en cas de conflicte es faran servir les definicions especificades més tard. Això permet ferfulls d'estils personalitzats per a algunes pàgines sense que es perdi l'aspecte

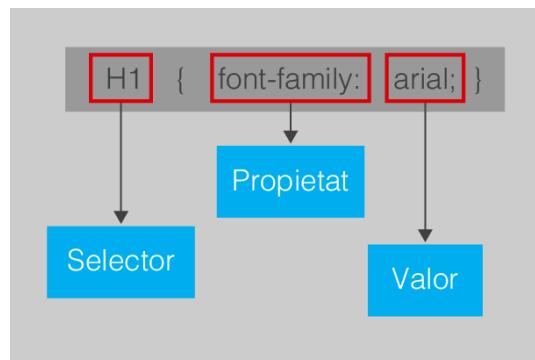
global d'una web.

```
1 <?xml-stylesheet href="empresa.css"?>
2 <?xml-stylesheet href="departament.css"?>
```

### 1.1.3 Regles CSS

Els documents de CSS normalment seran una llista de regles en què cada una té una forma semblant a la que es mostra en la figura 1.7.

**FIGURA 1.7.** Forma bàsica de les regles CSS



Els elements es van passant a CSS i aquest determina quines són les regles que se'ls ha d'aplicar a partir del selector. Per exemple, si es té el CSS següent quan s'està analitzant l'etiqueta *cognom* d'un document:

```
1 nom { color: red; }
2 cognom { color: blue; }
```

La regla de la primera línia fallarà, ja que no quadra el selector *nom* amb l'etiqueta *cognom* i per tant la regla s'ignorarà. En canvi, com que la segona regla sí que coincideix amb l'etiqueta aquesta sí que serà aplicada en la representació de l'etiqueta.

Cal tenir en compte que no s'aplica només a la primera regla, sinó a totes les regles que es puguin aplicar a l'element.

Un cop determinades les regles s'aplicaran els valors de les propietats que s'especifiquin en la **declaració** a l'element. En l'exemple anterior, per tant, es modificarà la propietat 'color' de l'etiqueta <cognom> amb el valor 'blue' que farà que el text es representi de color blau:

```
1 color:blue;
```

A pesar que en l'exemple anterior només es canvia una de les propietats de l'etiqueta, CSS permet que en una regla es puguin especificar tantes propietats com calgui, sempre que es vagin separant l'una de l'altra amb un punt i coma (;).

```

1 persona {
2     color: red;
3     margin: 5px;
4 }
```

## Selectors

Els selectors són un dels components bàsics de CSS, ja que serveixen per determinar quines regles s'han d'aplicar a cada element i quines no.

Per facilitar la tasca de seleccionar les etiquetes, CSS defineix diferents tipus de selectors:

- Universal
- D'etiquetes
- De fills
- De descendents
- De germans adjacents
- Per ID
- De classes
- D'atributs
- De pseudoclasses

### Selector universal

El selector universal es representa amb un '\*' i es fa servir per seleccionar tots els elements d'un document.

Això vol dir que es pot fer servir per definir característiques que seran aplicades a totes les etiquetes del document. Per exemple, amb això s'està definint que s'ha de pintar tot el text de color negre.

```

1 * { color: red; }
```

Per tant, aplicat al codi:

```

1 <alumnes>
2   <nom>Pere</nom>
3   <cognom>Garcia</cognom>
4 </alumnes>
```

S'obtindrà tot el text del document de color negre (figura 1.8).

**FIGURA 1.8.** Aplicació del selector universal

### Selector d'etiquetes

Es pot seleccionar un element específic simplement amb el nom. En aquest codi se seleccionen totes les etiquetes nom del document:

```
1 nom { color: red; }
2 cognom { color: blue; }
```

Que si s'aplica al codi següent

```
1 <classe>
2   <alumnes>
3     <alumne>
4       <nom>Pere</nom>
5       <cognom>Garcia</cognom>
6     </alumne>
7     <alumne>
8       <nom>Manel</nom>
9       <cognom>Puig</cognom>
10    </alumne>
11  </alumnes>
12 </classe>
```

Pintarà els <nom> vermells i els <cognom> blaus (figura 1.9).

**FIGURA 1.9.** Exemple de selector d'etiquetes

S'ha de tenir en compte que no solament se seleccionen els elements del mateix nivell, sinó tots els elements que tinguin el mateix nom d'etiqueta sense importar en quin nivell estiguin.

Si es volen aplicar les mateixes característiques a diversos elements, es poden especificar separat-los per comes. En l'exemple següent es pinta el text de <nom> i <cognom> de color vermell.

```
1 nom, cognom { color:red; }
```

Tampoc no seria cap error especificar-los per separat:

```
1 nom { color:red; }
2 cognom { color:red; }
```

## Selector de fills

El selector de fills es fa servir per seleccionar les etiquetes filles directes d'una altra. Es fa servir el símbol > per definir el pare i el fill.

Per exemple, si es parteix d'aquest XML

```

1 <classe>
2   <professor>
3     <nom>Marcel</nom>
4     <cognom>Puig</cognom>
5   </professor>
6   <alumnes>
7     <alumne>
8       <nom>Pere </nom>
9       <cognom>Garcia</cognom>
10    </alumne>
11    <alumne>
12      <nom>Joan</nom>
13      <cognom>Ferrarons</cognom>
14    </alumne>
15  </alumnes>
16 </classe>
```

I se li aplica aquest CSS:

```
1 alumne > nom { color: red; }
```

El resultat serà que només els elements <nom> descendents d'<alumne> són pintats de vermell (figura 1.10).

**FIGURA 1.10.** Selecció dels noms dels alumnes



No hi ha cap problema a fer servir altres selectors en combinació amb aquest. Per tant, si volem que es pintin de color vermell tant els noms com els cognoms dels alumnes es pot definir la regla d'aquesta manera:

```
1 alumne > nom, cognoms { color: red; }
```

El resultat serà el de la figura 1.11.

**FIGURA 1.11.** Selecció dels noms i cognoms dels alumnes



També es pot fer servir el selector d'etiquetes per triar diferents pares. Si es vol que els noms dels alumnes i dels professors siguin de color vermell simplement

se n'especifiquen les etiquetes una rere l'altra.

```
1 alumne, professor > nom { color: red; }
```

### Selector de descendents

Una possibilitat alternativa consisteix a definir que es vol que l'estil s'apliqui a algun dels descendents d'un element.

Per especificar que es vol fer servir aquest selector simplement es posen els noms dels elements l'un al costat de l'altre. Fent servir aquest selector s'aconsegueix pintar el cognom dels alumnes de color vermell i deixar el dels professors (figura 1.12).

```
1 alumnes cognom { color:red; }
```

**FIGURA 1.12.** Selecció de descendents



Es poden combinar amb el selector d'etiquetes cada un dels dos termes, de manera que amb el codi següent pintem el nom i cognom dels alumnes i dels professors de color vermell.

```
1 alumnes,professor nom,cognom { color: red; }
```

### Selector de germans adjacents

El selector de germans adjacents serveix per seleccionar elements que comparteixin el mateix pare i estiguin situats immediatament l'un rere l'altre.

En el codi XML següent:

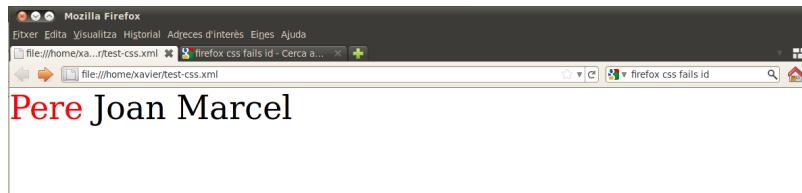
```
1 <classe>
2   <alumne>
3     <delegat/>
4     <nom>Pere</nom>
5   </alumne>
6   <alumne>
7     <nom>Joan</nom>
8   </alumne>
9   <alumne>
10    <nom>Marcel</nom>
11  </alumne>
12 </classe>
```

El CSS permet seleccionar aquests elements fent servir el símbol +. Per fer que el nom dels alumnes que estigui just darrere de l'atribut delegat siguin pintats de color vermell definirem la regla següent:

```
1 delegat+nom { color: red; }
```

El navegador només mostrarà de color vermell el text de l'etiqueta <nom> que sigui posterior a un element <delegat> (figura 1.13).

**FIGURA 1.13.** Selecció del germà de delegat



No es poden seleccionar elements que siguin de diferents nivells perquè no comparteixen el mateix pare. En el següent codi:

```
1 <classe>
2   <professor>
3     <nom>Marcel Puig</nom>
4   </professor>
5   <alumne>
6     <nom>Frederic Pi</nom>
7   </alumne>
8   <alumne>
9     <nom>Manel Puigdevall</nom>
10  </alumne>
11 </classe>
```

Aquest CSS no hi tindrà cap efecte a pesar de que els elements *alumne* i *nom* són adjacents:

```
1 alumne+nom { color: blue; }
```

## Selector per ID

L'atribut *id* es fa servir per poder identificar elements concrets dins d'un document XML o HTML. Per tant, l'atribut *id* només especificarà un sol element dins d'un document.

```
1 <classe>
2   <professor>
3     <nom>Marcel</nom>
4     <cognom>Puig</cognom>
5   </professor>
6   <alumnes>
7     <alumne id="delegat">
8       <nom>Pere </nom>
9       <cognom>Garcia</cognom>
10    </alumne>
11    <alumne>
12      <nom>Joan</nom>
13      <cognom>Ferrarons</cognom>
14    </alumne>
15  </alumnes>
16 </classe>
```

Es poden seleccionar els elements pel seu *id* fent servir el símbol #

Alguns navegadors encara fallen en seleccionar elements amb l'atribut `id` en documents XML.

```
1 alumne#delegat { color:red; }
```

Aquest codi pintarà de color vermell l'element que tingui l'atribut `id` amb el valor `delegat` (figura 1.14).

**FIGURA 1.14.** Selecció per l'identificador "delegat"



Això és molt usat en HTML però no sol funcionar en XML tret que tingui l'espai de noms d'XHTML definit.

### Selector de classes

Podria passar que, en algun document, a etiquetes que tenen el mateix nom a vegades els volem aplicar un estil i a vegades un altre.

Això se soluciona especificant l'atribut `class` en els elements, com en l'exemple següent:

```
1 <p class="normal">
2 Els fitxers de marques només són arxius de text que
3 es poden obrir amb un <span class="important">editor de textos</span>
4 senzill.
5 </p>
6
7 <p class="important">
8 Tot i que els editors especialitzats ens ajudaran a no
9 cometre errors.
10 </p>
11
12 <p class="normal">
13 Per tant, la creació de documents XML està
14 a l'abast de tothom.
15 </p>
```

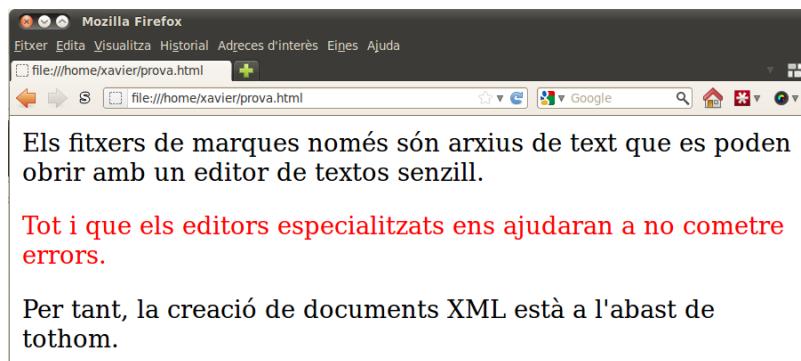
Es poden seleccionar els atributs per classes amb el nom de l'etiqueta, un punt i el nom de la classe:

```
1 p.important {
2   color:red;
3 }
```

Com que l'atribut `class` mateix es pot especificar en diferents elements, es pot fer una selecció de tots els atributs del mateix nom si no s'hi especifica el nom de l'element.

```
1 .important { color:red; }
```

Això donarà el que es mostra en la figura 1.15.

**FIGURA 1.15.** Selecció de tots els elements d'una classe

Els elements poden pertànyer a més d'una classe alhora. Simplement s'hi posen els diferents noms de classes separats per un espai.

```
1 <p class="important ressaltat">Oh!</p>
```

## Selector d'atributs

També hi ha un selector que permet triar les etiquetes en funció dels seus atributs.

```
1 <classe>
2   <alumnnes>
3     <alumne aprovat="si" delegat="si">
4       <nom >Pere </nom>
5       <cognom>Garcia</cognom>
6     </alumne>
7     <alumne>
8       <nom aprovat="si">Jordi</nom>
9       <cognom>Ferrarons</cognom>
10    </alumne>
11    <alumne aprovat="no">
12      <nom>Manel</nom>
13      <cognom>Puigdevall</cognom>
14    </alumne>
15    <alumne aprovat="si">
16      <nom>Frederic</nom>
17      <cognom>Pi</cognom>
18    </alumne>
19  </alumnnes>
20 </classe>
```

Es pot triar per si l'atribut existeix. Per exemple, per fer que el delegat sigui pintat de color verd (figura 1.16).

```
1 alumne[delegat] { color: green; }
```

**FIGURA 1.16.** A partir de l'atribut pinta de color verd el delegat del curs

També es pot seleccionar per a atributs que tinguin un determinat valor. Per exemple es poden marcar els aprovats i no aprovats de colors diferents:

```
1 alumne[aprovat="si"] { color:blue; }
2 alumne[aprovat="no"] { color:red; }
```

Que ens mostraria això (figura 1.17).

**FIGURA 1.17.** Diferenciar els aprovats dels no aprovats



O fins i tot seleccionar per diversos atributs, per aconseguir condicions més complexes. En l'exemple següent definim que si el delegat ha aprovat el pinta verd i si no ha aprovat el pinta groc:

```
1 alumne[aprovat="si"][delegat] { color: green; }
2 alumne[aprovat="no"][delegat] { color: yellow; }
```

En el cas que els valors de l'atribut siguin una llista es pot fer servir l'operador especial `~=` per seleccionar si algun dels valors és un valor en concret.

L'atribut `moduls` defineix tres valors per a l'element `alumne`:

```
1 <alumne moduls="XML BDD Programació"/>
```

Es poden seleccionar tots els elements `<alumne>` que tinguin el valor XML en l'atribut amb aquesta regla:

```
1 alumne[moduls]~="XML" { color: red; }
```

### Selectors de pseudoclasses

La idea d'aquests selectors és permetre seleccionar elements a partir d'una condició que no sigui el seu nom.

Les pseudoclasses se separen del nom de l'element al qual s'aplicaran amb dos punts (:).

Hi ha diverses pseudoclasses disponibles, que es poden veure en la taula 1.3.

**TAULA 1.3.** Pseudoclasses de CSS

Pseudoclasse	Significat
<code>first-child</code>	Permet seleccionar el primer element de la classe actual.
<code>link</code>	Selecciona l'element si és l'origen d'un enllaç ja visitat.
<code>visited</code>	Per als enllaços visitats d'un tipus determinat.

**TAULA 1.3** (continuació)

Pseudoclasse	Significat
active	Per als elements que s'estan activant (essent clicats, etc.).
linking	Per a enllaços en XLink.
hover	Elements que tenen el cursor al damunt.
focus	Tria l'element que té el focus.
lang	Permet triar els elements amb l'etiqueta especificant un idioma determinat.

Per tant, en aquest exemple:

```

1 <alumnes>
2   <nom>Frederic Pi</nom>
3   <nom>Pere Garcia</nom>
4   <nom>Manel Puigdevall</nom>
5 </alumnes>
```

Es poden definir característiques especials per al primer alumne de la llista fent servir una pseudoclasse:

```

1 alumnes:first-child {
2   color: red;
3 }
```

## Resolució de conflictes

Es pot donar el cas en què, en seleccionar les regles per aplicar a un element, una propietat estigui definida en diverses regles de les que se li han d'aplicar. I a més, que els valors d'aquesta propietat es contradiguin.

Quan a un element se li pot aplicar la mateixa propietat segons regles diferents i aquestes es contraduen, el CSS defineix que la regla per aplicar ha de ser la més específica.

Per exemple, com que les regles nom de l'exemple següent es contraduen s'agafarà la primera regla perquè és més concreta –defineix més clarament a quins elements s'està fent referència:

```

1 alumnes nom { color: red; }
2 nom { color: blue; }
```

Per tant, els noms sortiran de color vermell (figura 1.18).

**FIGURA 1.18.** En cas de conflicte es tria la regla més específica

Si hi ha regles que són igual d'específiques, aleshores **s'aplicarà la que s'hagi especificat més tard**. En l'exemple següent les regles són igual d'específiques i, per tant, es triarà la segona:

```
1 nom { color:red; }
2 nom { color:blue; }
```

O sigui, que pintarà el nom de color blau (figura 1.19).

**FIGURA 1.19.** En cas de conflicte es tria la regla més específica



## Declaracions

La declaració és el segon terme de les regles CSS i és la part que s'encarregarà de donar el format a les etiquetes. Es defineix entre claus “{ }” i estarà formada per una o més propietats. A cada propietat se li assignarà quin és el valor desitjat.

Per exemple, per fer que el text de l'atribut `alumne` sigui vermell es pot fer servir la propietat `color` i se li assigna el color amb la paraula “red” aprofitant que alguns dels colors es poden usar fent servir el nom en anglès:

```
1 alumne { color: red; }
```

Podem afegir tantes propietats com calguin en una regla. Es poden fer declaracions tan llargues com calgui:

```
1 alumne { color: red;
2         padding: 35px;
3         border-style: solid;
4         border-color: black;
5         border-width: 1px;
6 }
```

Per tant, coneixent les propietats de CSS es tindrà un potent mecanisme per representar les dades en documents XML i HTML.

## Herència

La primera lletra de CSS és de *cascading* ('cascada', en català), i indica que les propietats de les etiquetes no cal que es vagin repetint indefinidament, ja que s'hereten. Això vol dir que si apliquem unes propietats a un element, els seus elements fills automàticament les adquireixen.

Si tenim el document següent:

```
1 <alumnnes>
2   <alumne>
3     <nom>Frederic</nom>
```

```

4   <cognom>Pi</cognom>
5   </alumne>
6 </alumnes>
```

Si assignem el color vermell a l'element `<alumne>` automàticament els elements `<nom>` i `<cognom>` també adquiriran el color vermell.

```
1 alumne { color: red; }
```

Per evitar aquest comportament en algun dels descendents només cal sobreescrivir la propietat amb una regla nova. Per exemple, si volem `<cognom>` de color blau li assignem explícitament:

```

1 alumne { color: red; }
2 cognom { color: blue; }
```

Ara `<alumne>` rep per herència que ha de fer servir el color vermell, però com que ell té blau i aquesta és una regla més específica es pintarà blau (figura 1.20).

**FIGURA 1.20.** Es pot evitar l'herència de propietats sobreescrivint-les

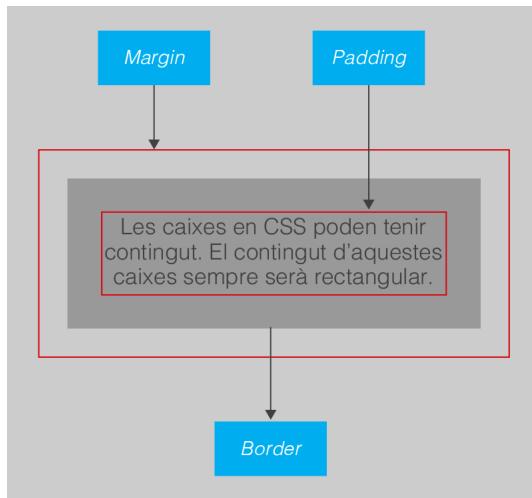


#### 1.1.4 Model de caixes

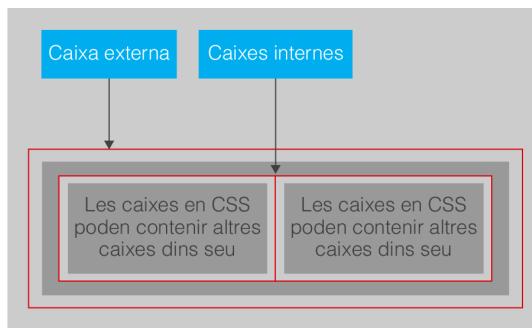
El CSS tracta tots els elements com si estiguessin dins d'una caixa rectangular que envolta el contingut (figura 1.21). En aquesta caixa es poden definir una sèrie de marges:

- **padding:** marge intern entre el contingut i la caixa.
- **margin:** marge entre la caixa i les altres caixes adjacents.

La caixa pot tenir una línia al seu voltant, que el CSS anomena **border**.

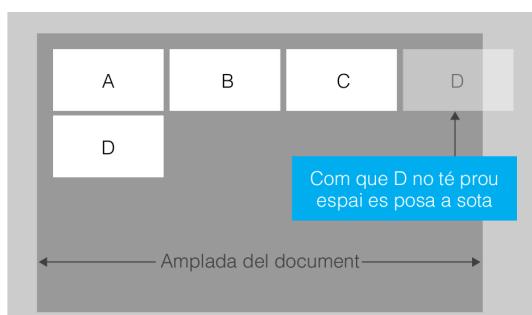
**FIGURA 1.21.** El contingut dels elements CSS fa servir el model de caixes

Però com que tant l'XML com l'HTML fan servir etiquetes niuades normalment tindrem caixes que estan dins d'altres caixes (figura 1.22).

**FIGURA 1.22.** El contingut dels elements CSS poden ser altres caixes

Per defecte:

- En XML quan es posicionen les caixes si no es diu el contrari sempre es col·loquen una al costat de l'altra fins que s'acaba l'espai horitzontal. Quan no queda més espai es representa la caixa següent a sota (figura 1.23)
- En HTML algunes etiquetes fan com les d'XML i d'altres no permeten tenir altres caixes al costat.

**FIGURA 1.23.** El posicionament per defecte col·loca les caixes horitzontalment

Es pot canviar el comportament per defecte de dues maneres:

- Amb la propietat `display`.
- Posicionant les caixes.

### Propietat `display`

Els elements tenen una propietat `display`, que entre d'altres coses determina com es posiciona la caixa respecte a les altres:

- `block`: treballa amb blocs de contingut. Fa que les caixes acabin com si tinguessin un salt de línia darrere seu. Per tant, es trenca amb el posicionament per defecte.
- `inline`: permet que les etiquetes flueixin amb la resta del contingut. Aquest és el comportament per defecte en representar contingut XML.

Per tant, si es té un codi XML com aquest:

```

1 <alumnes>
2   <nom>Pere</nom>
3   <nom>Frederic</nom>
4   <nom>Manel</nom>
5 </alumnes>
```

El resultat de mostrar-lo amb un navegador si no li definim cap estil serà posar-los `inline` com en la figura 1.24.

**FIGURA 1.24.** El document es presenta per defecte en 'inline'

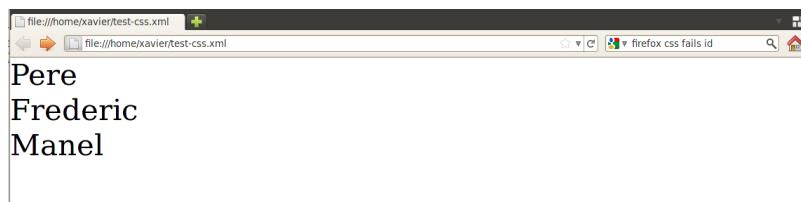


En canvi, si canviem la propietat `display` de l'element `<nom>` a `block`:

```
1 nom { display:block; }
```

Ens afegirà un salt de línia darrere de cada un dels elements com en la figura 1.25.

**FIGURA 1.25.** El resultat de mostrar les caixes amb 'block'



## Posicionament de caixes

El posicionament en CSS reposa sobre tres opcions:

- Normal
- Relatiu
- Flotant
- Absolut

### Posicionament normal

El posicionament normal és el posicionament per defecte i consisteix a anar posant les caixes una al costat de l'altra horitzontalment fins que no hi hagi espai per posar-ne una altra. Quan això passi les següents es posen al principi de la línia següent i tornem a començar

Una cosa que s'ha de tenir en compte és que **els marges horitzontals de les caixes es respecten i s'acumulen**. Per tant, dos elements alumne que tinguin la definició de format següent estaran separades 40 píxels:

```
1 alumne { margin: 20px; }
```

En canvi **verticalment els navegadors mesclen els marges**, de manera que no se sumen els marges sinó que **s'agafa el més gran**. Per tant, amb la mateixa definició anterior les caixes estaran separades només 20px verticalment.

### Posicionament relatiu

Si se segueix el posicionament normal les caixes ja tenen un lloc en el qual posicionar-se però es pot canviar aquest lloc convertint el posicionament de la caixa amb `position: relative`. Amb això s'aconsegueix que la caixa es mogui de la posició que li tocava indicant-li en quina quantitat.

Ens podem moure de les posicions normals amb `left`, `right`, `top`, `bottom`.

Per tant si partim del document següent:

```
1 <alumnnes>
2   <nom>Pere</nom>
3   <nom delegat="si">Frederic</nom>
4   <nom>Manel</nom>
5 </alumnnes>
```

Les dades de mesura que indiquem en el posicionament relatiu sempre es computen en relació amb la caixa anterior.

Si li afegim les regles següents per canviar-li la posició esquerra on ha de sortir l'element amb el `position: relative` i afegint 30 píxels a la posició esquerra:

```
1 nom { display:block; }
2 nom[delegat] {
3   position: relative;
4   left:30px;
5 }
```

En visualitzar el document es veurà que el posicionament de l'etiqueta ha canviat (figura 1.26).

**FIGURA 1.26.** Canvi en les posicions relatives d'un element



## Posicionament flotant

El posicionament flotant ens permet col·locar la caixa de manera que quedi surant en la part de la pantalla que li especifiquem.

La propietat que es fa servir per marcar posicionament com a flotant és `float`, que pot tenir els valors de la taula 1.4.

**TAULA 1.4.** Valors de la propietat "float"

Valor	Significat
<code>left</code>	La caixa es quedarà tant a l'esquerra com pugui.
<code>right</code>	La caixa es quedarà tant a la dreta com pugui.
<code>none</code>	La caixa no és flotant.
<code>inherit</code>	Ha de fer el mateix que faci el seu element pare.

Per tant, si es parteix d'aquest exemple:

```

1 <classe>
2   <professor>
3     <nom>Marcel Puig</nom>
4   </professor>
5   <alumnes>
6     <nom>Pere Gallerí</nom>
7     <nom delegat="si">Frederic Pi</nom>
8     <nom>Manel Fontcoberta</nom>
9     <nom>Llucia Martí</nom>
10   </alumnes>
11 </classe>
```

En aplicar-li les regles següents:

```

1 professor {
2   float: right;
3   width: 50%
4   color: red;
5 }
```

El nom del professor apareixerà a la dreta de la pantalla, mentre que els alumnes apareixeran normalment (figura 1.27).

**FIGURA 1.27.** El professor en ser "float" apareix a la dreta

Una de les característiques interessants de `float` és que fa que el contingut de les altres caixes aparegui al seu voltant.

Es pot donar el cas que sí que quedí sobre alguns elements posteriors si es dóna el cas que els elements del seu voltant siguin molt més petits que el `float`.

Per evitar que passi això es pot afegir un valor a l'atribut `clear` en els elements que no vulguem que siguin adjacents a una finestra `float` (taula 1.5).

**TAULA 1.5.** Valors de la propietat "clear"

Valor	Significat
<code>left</code>	El costat esquerre no pot ser adjacent a un <code>float</code> .
<code>right</code>	El costat dret no pot ser adjacent a un <code>float</code> .
<code>both</code>	No es permeten caixes flotants en cap dels seus costats.
<code>inherit</code>	Ha de fer el mateix que faci el seu element pare.
<code>none</code>	Ha de fer el funcionament per defecte.

Per tant, si s'afegeix `clear` a l'etiqueta `<nom>` de l'exemple anterior:

```

1 professor {
2     float: right;
3     width: 50%
4     color:red;
5 }
6 nom {
7     display:block;
8     clear: right;
9 }
```

El resultat serà que els alumnes no permeten `<professor>` a la seva esquerra, i per tant baixen un nivell (figura 1.28).

**FIGURA 1.28.** L'atribut "clear" impedeix que hi hagi "floats" a la dreta

## Posicionament absolut

El posicionament absolut serveix per posicionar exactament la caixa en un lloc determinat. Les caixes que es posicionen amb posicionament absolut surten del flux normal de representació.

Per fer posicionament absolut es fa servir l'atribut **position**, que només pot tenir dos valors: **absolute** i **fixed** (vegeu taula 1.6).

**TAULA 1.6.** Valors de l'atribut "position"

Valor	Funcionament	
<b>absolute</b>	Posiciona la caixa exactament en la posició especificada.	
<b>fixed</b>	La caixa es col·loca en una posició fixa de la pantalla. Això fa que no es mogui mai encara que el document pugui i baixi.	
		El posicionament <b>fixed</b> només té sentit per mostrar el document per pantalla. No té cap sentit per imprimir, etc.

Per tant, es pot posicionar un element en el lloc on es vulgui. Per exemple, a 200 píxels a l'esquerra i 20 des de dalt.

```

1 professor {
2   position:absolute;
3   left:200px; top:20px;
4   width: 100px;
5   height: 50px;
6   background-color:blue;
7 }
```

Aquest codi mostraria el que es veu en la figura 1.29.

**FIGURA 1.29.** L'element "professor" amb posicionament absolut



A diferència del posicionament relatiu, en aquest tipus de posicionament les mesures que definim són relatives al total de les mides de la pàgina, i no pas a algun dels elements que conté aquesta.

Les caixes amb posicionament absolut, en no estar dins del flux normal de representació, poden quedar per sota o per sobre de les altres (figura 1.30), i per aquest motiu se solen especificar amb l'amplada i alçada.

**FIGURA 1.30.** Problemes amb el posicionament absolut



### 1.1.5 Propietats

En cada una de les versions de CSS s'han anat afegint propietats noves que es poden aplicar a cada element.

Seria excessiu veure-les totes i, per tant, ens concentrarem en les més usades. De totes maneres es poden consultar totes les propietats disponibles en l'especificació oficial al web de W3C (<http://www.w3.org/TR/CSS21/propidx.html>).

Podem dividir les propietats en tres grans grups:

1. **Propietats de caixa:** permetran definir característiques de la representació de les caixes que contenen els elements.
2. **Propietats de text:** ens permetran definir com ha de ser representat el contingut textual del document.
3. **Propietats de color:** definiran el color en què s'han de representar els continguts i les caixes.

### Unitats

Algunes propietats necessitaran que els especifiquem un color o unes dimensions. En aquests casos, el CSS ofereix diverses alternatives per poder definir-hi un valor.

### Colors

Per especificar colors es poden fer servir diversos sistemes però el més corrent és fer-ho en RGB (*red, green, blue*), que consisteix a barrejar quantitats de cada color per representar-ne d'altres.

Això es pot fer amb la funció `rgb()`:

```

1 nom {
2   color: rgb(250,20,10);
3 }
4 cognom{
5   color: rgb(80%, 40%, 0%);
6 }
```

O bé el seu valor en *hexadecimal*

```

1 nom {
2   color: #cc6600;
3 }
```

També hi ha una sèrie de colors predefinits que es poden usar fent servir la paraula en anglès: red, blue, yellow, green, black, white...

```

1 alumne {
2   color:red;
3 }
```

## Mides

Quan alguna propietat demana que se li especifiqui una mida es poden especificar de dues maneres diferents:

- **Valors absoluts**
- **Valors relatius**

Les mides en **valors absoluts** són aquelles en les quals s'especifica la mida exacta en la qual es vol que es representi aquella propietat. És corrent definir-ho en píxels (px) però també es poden fer servir una sèrie de valors de text (small, medium, large... )

```

1 nom {
2     width: 50px;
3 }
4
5 cognom{
6     width: large;
7 }
```

També es poden especificar les mides en **valors relatius**. En aquest cas s'expressa l'increment o decrement de la mida per defecte del tipus de lletra de la caixa. Podem definir aquestes mides en percentatge (%) o en escalat (em).

```

1 alumne {
2     width: 100px;
3 }
4
5 nom {
6     width: 50%;
7 }
8
9 cognom
{
10    width: 1.2em;
11 }
12 }
```

### Unitats 'em'

Les unitats 'em' fan referència a l'increment o decrement sobre la mida de la font que s'està fent servir. Tenint en compte que la mida normal és *1em* si s'especifica *2 em* s'està dient que les lletres s'han de representar al doble de la mida normal. També es pot fer servir per reduir-ne la mida a la meitat amb *0.5em*

## Propietats de caixa

El CSS representa totes les marques com a caixes rectangulars. Per defecte les caixes tenen unes propietats per defecte però es poden canviar simplement redefinint-les. Es poden veure algunes de les propietats en la taula 1.7.

**TAULA 1.7.** Algunes de les possibles propietats de caixa

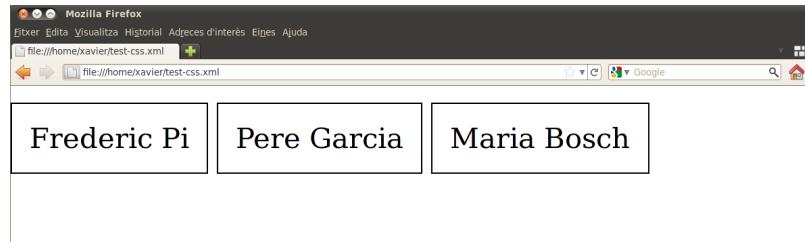
Propietat	Ús
margin	Serveix per definir quin és l'espai entre caixes. Per defecte és 0.
padding	Permet definir el marge intern d'una caixa. Per defecte és 0.
border	Permet definir si s'ha de representar la línia que envolta la caixa. Per defecte no es fa.
width	Permet definir l'amplada d'una caixa.
height	Permet definir l'alçada d'una caixa.

Les propietats `margin`, `padding` i `border` especificades directament assignen els seus valors als quatre costats de la caixa. D'aquesta manera, amb aquest codi:

```
1 nom {  
2   border: 1px black solid;  
3   padding: 10px;  
4 }
```

El resultat serà que sortirà una línia que envoltarà tota la caixa i el contingut tindrà un marge intern de 10 píxels (figura 1.31).

**FIGURA 1.31.** Canvi en el marge intern i una línia envolant



Es poden definir els valors dels marges d'un costat en concret especificant els valors de cada costat (dalt, dreta, baix, esquerra: sempre en aquest ordre, com les agulles del rellotge).

```
1 nom {  
2   padding: 30px 50px 1px 1px;  
3 }
```

O bé fent servir la versió de l'atribut que permet posar explícitament els valors:

```
1 nom {  
2   padding-left: 1px;  
3   padding-top: 30px;  
4   padding-right: 50px;  
5   padding-bottom: 1px;  
6 }
```

Les darreres regles ens mostraran la figura 1.32.

**FIGURA 1.32.** Canvis en els marges interns dels elements



Una cosa semblant passa amb la propietat `border`, que permet especificar el gruix de la línia, el color i el tipus de línia que es vol.

```
1 nom { border: 1px black solid; }
```

També es poden especificar les línies de cada un dels costats de manera individual:

```

1 nom {
2   border-top: 2px red solid;
3   border-bottom: 2px blue dashed;
4   border-left: 1px green dotted;
5   border-right: 1px yellow outset;
6 }

```

Que mostrerà una imatge com la de la figura 1.33.

**FIGURA 1.33.** Aplicar propietats diferents per a cada línia



També es disposa de propietats per canviar només característiques concretes de la línia, com ara:

```

1 nom {
2   border-width: 1px;
3   border-color: black;
4   border-style: solid;
5   border-top-color: red;
6 }

```

## Propietats de text

Un dels aspectes fonamentals a l'hora de donar format serà poder-ho fer al contingut de text dels documents. En CSS això es fa de dues maneres:

- **Propietats de tipus:** fan referència a com es veuran les lletres
- **Propietats text:** fan referència a com es mostrerà el bloc de text en conjunt

## Propietats de tipus

Aquestes propietats permetran canviar tot el que fa referència al tipus de lletra que es fa servir per mostrar el text. Es poden canviar la majoria dels paràmetres de cop fent servir la propietat `font`.

```

1 nom {
2   font:italic bold 12px Verdana, Geneva, Arial, sans-serif;
3 }

```

També es pot canviar només algun aspecte concret amb propietats més específiques com les de la taula 1.8

**TAULA 1.8.** Propietats habituals del tipus "font"

Propietat	Ús
<code>font-family</code>	Permet especificar quin tipus de lletra fem servir i la família a la qual pertany. Normalment se n'hi posen diverses per si l'equip no té instal·lat el tipus de lletra demandat.
<code>font-size</code>	Serveix per definir la mida del tipus de lletra que es fa servir.
<code>font-weight</code>	Defineix el pes de la lletra. Generalment es fan servir negretes ( <code>bold</code> ) o rodones ( <code>normal</code> ), però hi ha més valors possibles.
<code>font-style</code>	Permet definir l'estil de lletra. Els valors més corrents són cursiva ( <code>italic</code> ) o obliqua ( <code>oblique</code> ).
<code>font-variant</code>	Sobretot es fa servir per convertir el contingut a versaletes ( <code>small-caps</code> ).

### Famílies de tipus de lletra

Una família de tipus de lletra és un grup de tipus de lletra que comparteixen característiques comunes. En CSS se'n defineixen cinc famílies, que contenen un gran nombre de tipus:

- **sans-serif**: tipus de lletra de pal sec. Per exemple: Arial, Verdana, Helvetica...
- **serif**: són els que contenen gràcies, que són unes línies de decoració en els extrems d'algunes lletres. Per exemple: Times, Georgia, Times New Roman...
- **monospace**: tipus de lletra de mida fixa. Tots els caràcters ocupen el mateix espai. Per exemple: Courier.
- **cursive**: en aquesta família les lletres semblen escrites a mà. Per exemple: Comic Sans...
- **fantasy**: la família està formada per tipus de lletres en què les lletres tenen decoració afegida. Per exemple: Impact.

En l'XML següent...

```

1 <alumnes>
2   <alumne>
3     <nom>Frederic</nom>
4     <cognom>Pi</cognom>
5   </alumne>
6   <alumne>
7     <nom>Pere</nom>
8     <cognom>Garcia</cognom>
9   </alumne>
10 </alumnes>
```

apliquem aquest CSS:

```

1 alumne {
2   display:block;
3 }
4
5 nom {
6   font:italic bold 30px Georgia, serif;
7 }
8
9 cognom {
10   font-family:Times,serif;
11   font-size:15px;
12 }
```

Observeu que els tipus de lletra han canviat (figura 1.34).

**FIGURA 1.34.** Canvis en els tipus de lletra



## Propietats de text

A més de les propietats de les lletres també es poden canviar les propietats de com es visualitzarà el text. Podem canviar l'orientació, el color, l'espaiat entre línies, etc. (taula 1.9).

**TAULA 1.9.** Propietats del text

Propietat	Ús
word-spacing	Espaiat entre paraules.
letter-spacing	Espaiat entre lletres.
text-decoration	Permet decorar el text amb subratllats o línies diverses.
vertical-align	Alineació vertical del contingut dins de la caixa.
text-align	Alineació del text en la caixa.
line-height	Alçada de la línia de text.
color	Color amb què es mostrerà el text.

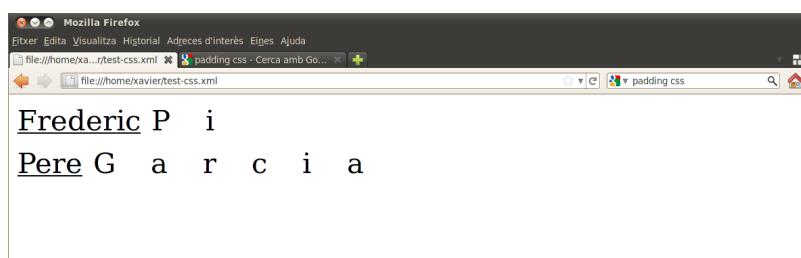
Les regles següents:

```

1 nom {
2   text-decoration:underline;
3 }
4
5 cognom {
6   letter-spacing:20px;
7 }
```

Generaran un resultat com el que es veu a la figura 1.35.

**FIGURA 1.35.** Canvi de la representació del text



## Propietats de color i dimatge

Una altra de les característiques del model de caixes és que en el fons de les caixes es pot definir un color o una imatge

El color de fons es defineix amb **background-color** i s'hi especifica el color que es vol:

```
1 nom {  
2   background-color: blue;  
3 }
```

Les imatges s'afegeixen passant l'URL de la imatge.

```
1 nom {  
2   background-image: url('fons.png');  
3 }
```

El funcionament per defecte és omplir el fons repetint la imatge tantes vegades com calgui fins a cobrir-lo tot, tal com es mostra en la figura 1.36.

**FIGURA 1.36.** Imatge de fons



Hi ha una sèrie de propietats que permeten canviar aquest comportament, que podeu veure en la taula 1.10.

**TAULA 1.10.** Propietats del fons de caixa

Propietat	Ús
<code>background-repeat</code>	Defineix de quina manera s'ha de repetir la imatge, si s'ha de fer.
<code>background-position</code>	Es fa servir per posicionar la imatge en un punt concret.
<code>background-attachment</code>	Defineix si el fons es mourà en desplaçar la imatge o no.

### 1.1.6 Amagar contingut

A vegades pot interessar que no tot el contingut del document XML sigui mostrat, ja sigui perquè no aporta res a un lector humà o per altres raons.

Això es pot fer mitjançant dues propietats:

- visibility
- display:none

## Propietat 'visibility'

La propietat `visibility` permet fer que algun element no es mostri quan es defineix en un determinat element.

Si partim del document següent:

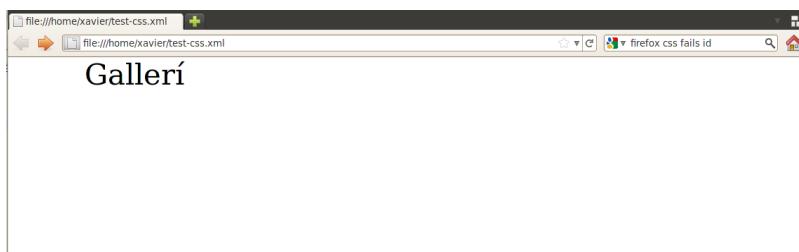
```
1 <persona>
2   <nom>Pere</nom>
3   <cognom>Gallerí</cognom>
4 </persona>
```

Podem fer que no es mostri l'element `<nom>` definint la propietat `hidden`:

```
1 persona {
2   display:block;
3 }
4
5 nom {
6   visibility:hidden;
7 }
```

En la figura 1.37 podeu comprovar que, en efecte, no es mostra el nom si aquest té el valor `visibility:hidden`.

**FIGURA 1.37.** Aplicar "visibility:hidden"



És important veure que, a pesar que no se'n mostra el contingut, encara s'està reservant l'espai que ocuparia.

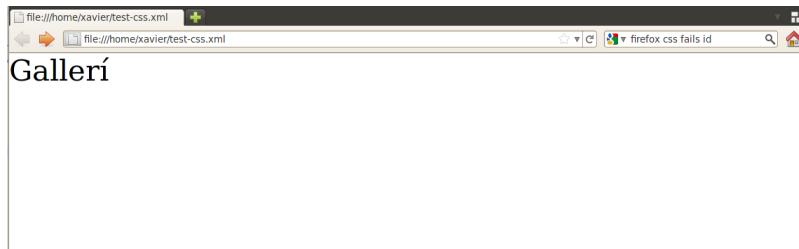
## Propietat display:none

Amb aquesta propietat el que s'aconsegueix és que l'etiqueta no sigui mostrada. Prenent el codi de l'exemple anterior, amb el CSS següent:

```
1 persona {
2   display:block;
3 }
4
5 nom {
6   display:none;
7 }
```

S'aconsegueix que l'etiqueta <nom> no es mostri ni, a diferència del que passa amb `visibility:hidden`, es reservi cap espai (figura 1.38).

**FIGURA 1.38.** El nom desapareix de la presentació



### 1.1.7 Afegir contingut a l'XML

Hi ha tota una sèrie d'aspectes que són molt diferents en XML i HTML. Per exemple l'HTML té etiquetes que s'encarreguen de mostrar el contingut en taules, llistes numerades, etc.

En XML totes aquestes ajudes per a la representació de la informació no existeixen i, per tant, s'haurà de buscar alguna alternativa per representar-les.

#### Crear taules XML

En HTML es pot fer servir <table> per fer taules però en XML no existeix aquesta etiqueta. No existeix perquè els documents XML no especificuen de quina manera s'han de mostrar les dades.

Es pot simular la manera de mostrar les taules per pantalla fent combinacions de posicionament però això representa molta feina.

L'atribut `display` de CSS permet crear taules des del full d'estil amb les etiquetes de la taula `???`. Això fa que es pugui associar cada una de les etiquetes del document a files, cel·les, etc.

**TAULA 1.11.** Taules en CSS

Valor	Descripció
<code>display:table</code>	Defineix que comença una taula.
<code>display:table-row</code>	Defineix l'etiqueta com una línia de la taula.
<code>display:table-cell</code>	L'etiqueta amb aquest valor serà una cel·la.

Si partim del document següent:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <classe>
3      <alumnes>
4          <alumne>
5              <nom>Pere</nom>
6              <cognom>Punyetes</cognom>
7          </alumne>

```

```

8   <alumne>
9     <nom>Filomeno</nom>
10    <cognom>Garcia</cognom>
11  </alumne>
12  <alumne>
13    <nom>Mariano</nom>
14    <cognom>Puigdevall</cognom>
15  </alumne>
16 </alumnes>
17 </classe>
```

Es pot veure que la informació pot ser mostrada en forma de taules; si definim que la taula comença a <alumnes>, cada fila està representada per <alumne> i el contingut de cada cel·la seran les etiquetes <nom> i <cognom>:

```

1  alumnes {
2    display:block;
3    display:table;
4    margin:10px;
5  }
6
7  alumne {
8    display:table-row;
9  }
10
11 nom, cognom {
12   display:table-cell;
13   border: 1px solid #000000;
14   padding: 3px;
15   border-spacing: 0px;
16 }
17
18 nom { background-color:blue; }
19 cognom {background-color:red; }
```

Amb la decoració afegida el resultat serà semblant al que es pot veure en la figura 1.39.

**FIGURA 1.39.** Presentació de les dades XML en taules

Pere	Punyetes
Filomeno	Garcia
Mariano	Puigdevall

## Llistes de valors

A partir de determinades etiquetes es poden crear llistes de resultats fent servir la propietat `display:list-item`. Normalment aquests elements es mostren amb un cercle davant del seu valor

Per exemple, en aquest document

```

1 <persones>
2   <persona>
3     <nom>Pere Gallerí</nom>
4   </persona>
5   <persona>
```

S'ha de vigilar de posar marge a la caixa que tindrà la llista perquè si no pot passar que quedi fora de pantalla.

```

6      <nom>Frederic Pi</nom>
7      </persona>
8  </persones>

```

Es pot fer que es mostrin les <persones> en forma de llista amb un codi CSS com el següent:

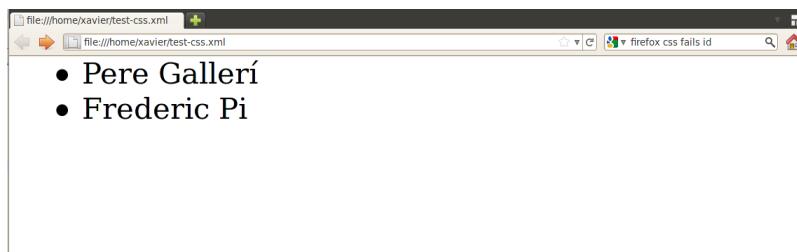
```

1 persona {
2   display:block;
3   margin-left:40px;
4
5 }
6 persona {display:list-item;  }

```

Es mostrerà el document tal com apareix en la figura 1.40.

**FIGURA 1.40.** Representació en forma de llista



Per representar una llista es poden fer servir diferents propietats (taula 1.12) que permeten personalitzar-les tal com calgui.

**TAULA 1.12.** Representar una llista amb CSS

Propietat	Significat
<code>list-style-type</code>	Permet definir de quin tipus serà la llista. Pot tenir una sèrie de valors numèrics o símbols. Per exemple: <code>decimal</code> , <code>circle</code> , <code>square</code> , <code>disc</code> ...
<code>list-style-position</code>	Defineix si les marques han d'aparèixer abans o en el flux de text.
<code>list-style-image</code>	Permet definir una imatge com a marca de la llista.

Per exemple, podem canviar el símbol d'ítem de llista per una imatge determinada.

```

1 persona {
2   display:block;
3   margin-left:40px;
4 }
5
6 persona
7 {
8   display:list-item;
9   list-style-image: url("bola.png");
10 }

```

## Imatges en XML

En els documents de marques les imatges no s'afegeixen sinó que s'hi enllacen.

El problema és que l'XML no té cap etiqueta que serveixi per mostrar imatges, ni té cap manera de fer enllaços. Per tant, l'única manera que hi ha és per mitjà d'un llenguatge XML anomenat *XLink*.

Una opció alternativa consisteix a afegir les imatges per mitjà de la propietat *background* de les cel·les de CSS. Amb aquesta propietat podrem fer que aparegui una imatge en representar una etiqueta d'un document XML.

```

1 logotip {
2   display:block;
3   background: url(logotip.png);
4   background-repeat: no-repeat;
5 }
```

### XLink

L'XLink és un llenguatge basat en XML que està pensat per afegir enllaços als documents XML; en podeu trobar més informació en el web <http://www.w3.org/TR/xlink/>.

## Afegir text que no es troba en el document

Es pot afegir text nou en mostrar un document XML fent servir els pseudoselectors. Els pseudoselectors permetran afegir informació complementària a les etiquetes.

Els pseudoselectors més usats són els que es mostren a la taula 1.13 :

**TAULA 1.13.** Pseudo-selectors

pseudo-selector	Descripció
:first-letter	Permet definir estils especials a la primera lletra del contingut seleccionat pel selector.
:first-line	Es fa servir per donar format a la primera línia d'un contingut de dades de bloc.
:before	Permet afegir contingut abans d'un element.
:after	Permet afegir contingut després d'un element.

No tots els pseudoselectors funcionen amb XML.

Gràcies als pseudoselectors es pot afegir contingut a la representació fent servir la propietat *content*. Per exemple, en l'XML següent:

```

1 <persones>
2   <professor>
3     <nom>Pere Gallerí</nom>
4   </professor>
5   <alumnes>
6     <nom>Frederic Pi</nom>
7     <nom>Filomena Garcia</nom>
8     <nom>Manel Puigdevall</nom>
9   </alumnes>
10 </persones>
```

es pot afegir el text “professor:” davant del nom del professor amb la regla *:before*:

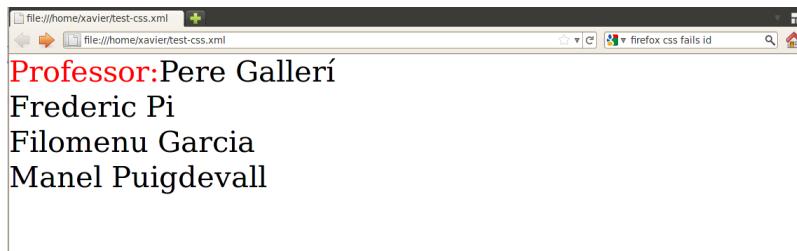
```

1 professor nom:before {
2   content:"Professor:";
3   color: red;
4 }
5 nom {display:block;}
```

I mostrerà el text “Professor:” abans del nom del professor (figura 1.41).

text “professor”

**FIGURA 1.41.** Representació del document XML amb el



### Mostrar el valor dels atributs

El CSS no té cap manera senzilla de mostrar les dades que hi ha en els atributs. Fent servir els pseudoselectors es podran mostrar els valors dels atributs fent servir la funció `attr()`.

En l'exemple següent els càrrecs es defineixen fent servir un atribut:

```

1 <alumnes>
2   <nom>Frederic Pi</nom>
3   <nom carrec="delegat">Pere Garcia</nom>
4   <nom>Manel Puigdevall</nom>
5   <nom carrec="subdelegat">Maria Bosch</nom>
6 </alumnes>
```

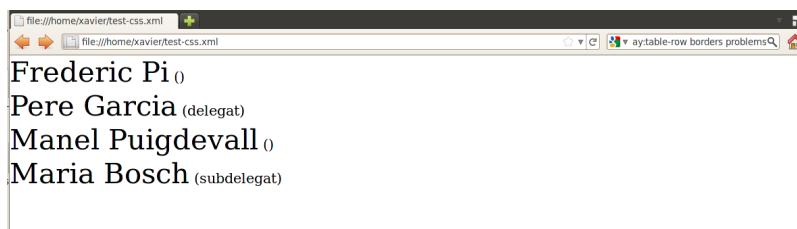
Si es vol mostrar la llista d'alumnes però que al costat surti el càrrec ho podem fer de la manera següent:

```

1 alumnes {
2   display:block;
3 }
4
5 nom:after {
6   content: "(" attr(carrec) ")";
7 }
```

i mostràrà el de la figura figura 1.42.

**FIGURA 1.42.** Mostrar l'atribut d'un document XML



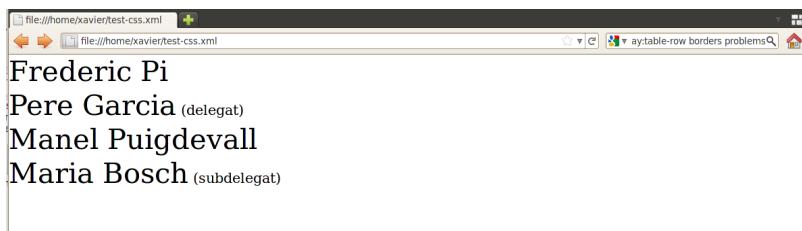
Com que alguns elements no tenen l'atribut es mostren els parèntesis () sense contingut a dins. Això es pot solucionar posant el pseudoselector només pels elements que tenen l'atribut:

```

1 nom[carrec]:after
2 {
3   font-size:8px;
4   content: " (" attr(carrec) ")";
5 }
```

El resultat final serà el que es mostra en la figura 1.43.

**FIGURA 1.43.** Mostrar només els atributs dels elements que en tenen



## 1.1.8 Validació

Generalment els navegadors web que es troben amb una línia que conté algun error (una propietat mal escrita, unitats mal definides, etc...) o que no entenen (perquè no l'han implementat) la solen ignorar.

En conseqüència, una de les coses importants per no tenir problemes de representació del CSS és que el document segueixi tant com pugui els estàndards.

**FIGURA 1.44.** Pàgina del W3C per validar CSS

The screenshot shows the W3C CSS Validation Service homepage. At the top, there is a navigation bar with links to various languages. Below the navigation bar, the main heading is 'CSS Validation Service' with the subtitle 'Check Cascading Style Sheets (CSS) and (X)HTML documents with style sheets'. There are three input options: 'By URI', 'By file upload', and 'By direct input'. The 'By URI' option is selected. A form field labeled 'Address:' contains a placeholder 'Enter the URI of a document (HTML with CSS or CSS only) you would like validated:'. Below the form is a 'Check' button. Further down the page, there is a note about Mozilla Foundation support and a 'Donate' button. At the bottom, there is a note about validating CSS in HTML documents and a footer with links to 'About', 'Documentation', 'Download', 'Feedback', and 'Credits'. The footer also includes the W3C Quality Assurance logo and a copyright notice.

Igual que en molts llenguatges de marques, podem validar el CSS per comprovar que no hi ha cap error i que no s'incompleix cap norma de l'especificació.

Els validadors de CSS ens permetran comprovar que el document de regles que estem definint no incompleix cap estàndard ni conté errors.

Hi ha diverses maneres de validar documents CSS però una de molt popular és el validador en línia del W3C <http://jigsaw.w3.org/css-validator/> (figura 1.44).

## 1.2 HTML / XHTML

A pesar que es pot representar la informació fent servir XML i CSS, el més habitual sol ser representar-la en algun tipus de llenguatge de marques que estigui pensat per a la representació. El llenguatge de marques per excel·lència a l'hora de presentar la informació és l'HTML (*hypertext markup language*).

L'HTML va sorgir durant el 1989 quan Tim Berners-Lee va proposar un sistema d'hipertext per compartir documents científics per mitjà d'Internet perquè poguessin ser visualitzats des de diferents sistemes.

L'HTML és un llenguatge de marques que permet publicar informació molt diversa pensant en l'estructura dels documents i en com es representaran les dades que contenen. Ha tingut tant d'èxit que pràcticament s'ha convertit en una forma de comunicació universal.

L'HTML és una recomanació del W3C. El gran èxit que ha tingut ha provocat que n'hagin sortit diverses versions al llarg dels anys per intentar adaptar-se a les noves demandes dels usuaris.

### HTML5

Des de la publicació de l'HTML 4.01 (1999), l'activitat de millora de l'HTML es van aturar perquè el W3C es va centrar en l'XHTML, ja que la intenció era que l'XHTML substituís completament l'HTML.

Com a resposta a la lentitud en els canvis en l'HTML, una sèrie d'empreses pel seu compte (Mozilla, Apple i Opera) van crear el WHATWG (Web Hypertext Application Technology Working Group), que es va centrar en crear l'HTML5. Des del 2007 el W3C torna a definir les recomanacions d'HTML.

L'objectiu fonamental de l'HTML5 és suportar les darreres tecnologies multimèdia mentre es manté el llenguatge de marques fàcil de llegir per als humans i fàcil d'entendre per als programes i dispositius. En podeu trobar més informació en els enllaços següents:

- <http://www.w3.org/TR/html5/>

- <http://whatwg.org/html>

## 1.2.1 HTML

L'HTML està pensat per definir l'estructura d'un document de text a partir d'una sèrie d'etiquetes predefinides formades per un nom envoltat dels símbols < i >. Cada una de les etiquetes servirà per definir l'estructura d'un document a part d'aportar-li informació semàntica sobre el contingut respecte al document.

En les primeres versions, algunes de les etiquetes i propietats estaven pensades també per marcar quina seria la manera en què es representaria la informació, però actualment s'estan eliminant de l'estàndard.

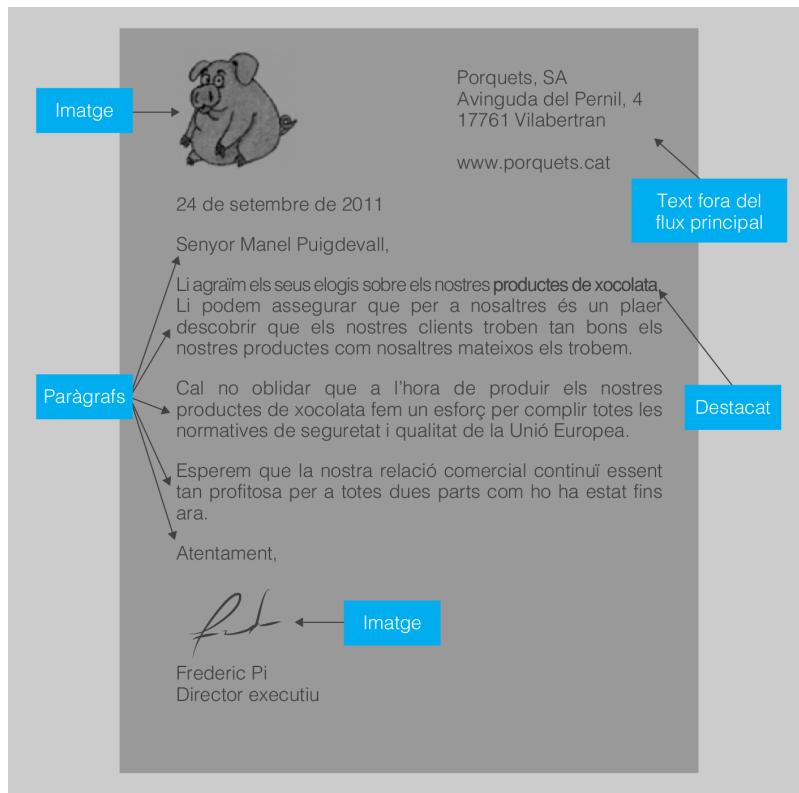
Actualment la creació d'una pàgina web consistirà a definir l'estructura del document per mitjà d'HTML i definir-ne el format per mitjà dels fulls d'estil CSS.

La separació de la informació de l'estructura de les dades és un dels components clau per donar dinamisme a les pàgines web.

### Definició de l'estructura d'un document

Si s'analitza un document de text a grans trets podem veure que té una estructura definida. Primer hi sol haver títols, text repartit en paràgrafs, imatges... (figura 1.45).

**FIGURA 1.45.** Determinació de l'estructura d'un text



Aquestes parts són fàcils de detectar per una persona però no és igual de senzill que un programa ho pugui fer. L'HTML intenta definir l'estructura d'un document de manera que sigui senzill per a un programa interpretar què són cada una de les dades que va trobant. Per fer-ho fa servir tota una sèrie d'etiquetes que indicaran quin és el paper que tenen cada una de les dades en el document.

S'han definit etiquetes per marcar cada una de les seccions en què podem definir un document.

## Títols

Es poden definir **títols** fent servir les etiquetes `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>` i `<h6>`. Aquestes etiquetes serveixen per definir diferents nivells de títols en un document.

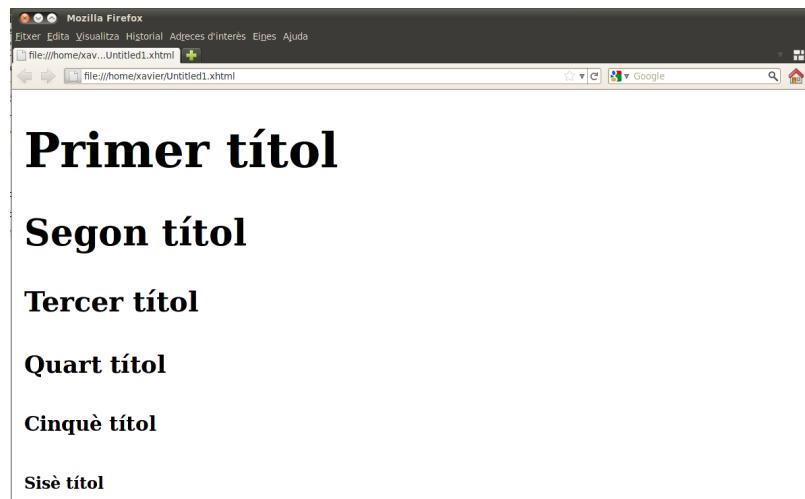
Els navegadors, en carregar un document HTML, ja “comprenen” per defecte quin és el significat dels diferents nivells de títols i els representen convenientment: amb mides diferents i amb un salt de línia després (figura 1.46).

```

1 <h1>Primer títol</h1>
2 <h2>Segon títol</h2>
3 <h3>Tercer títol</h3>
4 <h4>Quart títol</h4>
5 <h5>Cinquè títol</h5>
6 <h6>Sisè títol</h6>

```

**FIGURA 1.46.** Representació per defecte dels títols en el Firefox



## Paràgrafs

Una de les parts bàsiques que formen un document solen ser els **paràgrafs**, que en HTML es defineixen fent servir l'etiqueta `<p>`.

Per defecte els paràgrafs acaben amb un salt de línia encara que no s'indiqui explícitament en el document (figura 1.47).

```

1 <h1>Primer títol</h1>
2 <p>Aquest és el primer paràgraf</p><p>Segon paràgraf</p>

```

**FIGURA 1.47.** Representació dels paràgrafs en el Firefox

## Agrupació de contingut

Gairebé sempre, els títols i un nombre determinat de paràgrafs d'un text estan lligats per un contingut semàntic comú. Però com que la representació de les dades pot reordenar el contingut del document cal algun sistema per mantenir aquestes parts unides.

L'HTML permet agrupar continguts que tenen algun tipus de lligam per mitjà de l'etiqueta <div>.

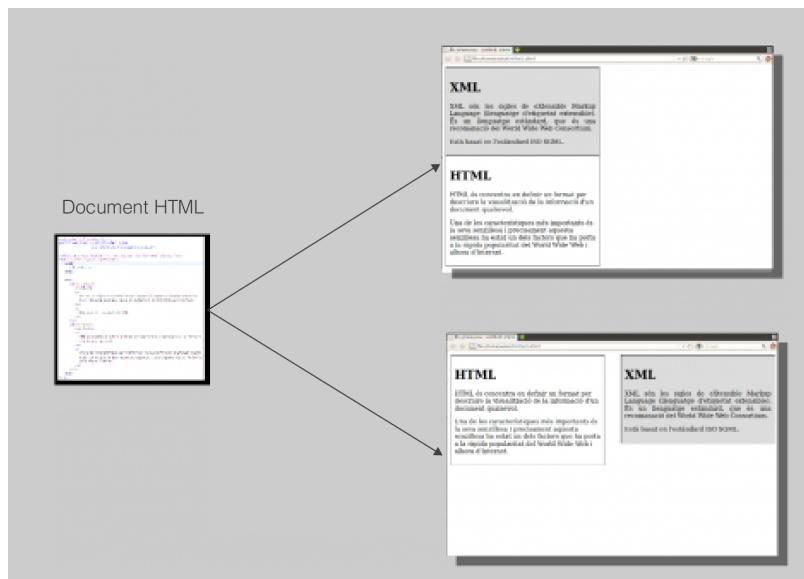
```

1 <div>
2   <h1>Titol 1</h1>
3   <p>Contingut de text</p>
4   <p>Més contingut de text</p>
5 </div>

```

Generalment els elements <div> se solen identificar per mitjà d'atributs id o class perquè el full d'estil hi pugui fer referència de manera més senzilla.

Un cop es té el contingut agrupat convenientment es farà més fàcil per alsfulls d'estil presentar el document reordenant els continguts d'un document sense que se'n perdi la coherència interna (figura 1.48).

**FIGURA 1.48.** Amb "div" es pot reordenar coherentment

## Ressaltar text

Hi ha tota una sèrie d'etiquetes més que tenen diverses funcions per a parts concretes del text:

- Definir text com a citacions: <blockquote>, <q> i <cite>.
- Marcar que s'ha de donar èmfasi a una part del text: <em> i <strong>
- Definir contingut com abreviacions, acrònims i definicions: <abbr>, <acronym>, <dfn>
- Permetre l'entrada de text preformatat: <pre> i <code>

Però moltes d'aquestes etiquetes poden ser substituïdes per l'etiqueta <span>. L'element <span> aporta una manera de separar lògicament contingut dins d'un paràgraf. Gràcies a aquesta separació es podrà donar un format diferent a parts del seu contingut.

Per defecte els navegadors representen les etiquetes <span> com a text normal, de manera que si volem que el text tingui un format diferent del normal se li haurà d'assignar per mitjà d'un full d'estil. És per aquest motiu que gairebé sempre s'hi afegeix un atribut `id` o `class`.

**FIGURA 1.49.** El paràgraf té dues parts amb format diferent

### Cicles formatius de formació professional

dimarts, 12 d'abril de 2011 12:23

Els cicles formatius són els ensenyaments que preparen per a l'exercici d'una professió determinada; s'agrupen en famílies professionals i poden ser de **grau mitjà** o de **grau superior**.

Per exemple, si es volgués mostrar el text tal com apareix a la figura 1.49, es podria definir la part que ha de ser en negreta envoltant-la amb <span> d'aquesta manera:

```

1  <h1>Cicles formatius de formació professional</h1>
2  <p>dimarts, 12 d'abril de 2011 12:23</p>
3  <p>Els cicles formatius són els ensenyaments que preparen per a l'exercici d'
   una professió determinada;
4  s'agrupen en famílies professionals i poden ser de <span class="negreta">grau
   mitjà</span> o de <span class="negreta">grau superior</span>.
5  </p>

```

I posteriorment, en el full d'estil, definint que es vol negreta en la classe `negreta`.

## Imatges

Les imatges són un dels elements importants en les pàgines web. Podem classificar les imatges que trobem en una pàgina web en dos grans grups:

- **Imatges d'adornament:** no són essencials per al contingut de la pàgina, simplement hi són per millorar-ne la presentació. Per tant, s'haurien de carregar des del CSS.

- **Imatges de contingut:** formen part del contingut de la pàgina. Per tant, s'han de definir en el document.

És important saber que les imatges no s'afegeixen realment al document sinó que en aquest només s'hi desa una referència al lloc on les podem trobar via una URL.

Per definir imatges en un document es fa servir l'element <img>, que té dos atributs obligatoris:

- L'atribut **src**, en què s'especificarà l'URL de la imatge.
- L'atribut **alt**, que contindrà un text que visualitzaran els programes que accedeixin al document HTML però no tinguin capacitat gràfica per mostrar imatges.

En HTML l'etiqueta <img> no cal que sigui tancada però en XHTML ho ha d'estar per força.

Per tant, es pot fer que aparegui una imatge en qualsevol document HTML definint la imatge amb una línia com aquesta:

<sup>1</sup> 

## Llistes

Les llistes són una altra manera de representar la informació en HTML. Hi ha tres tipus de llistes disponibles (figura 1.50):

- Llistes numerades
- Llistes no numerades
- Llistes de definicions

Les **llistes numerades** permeten especificar una sèrie de valors precedits per un nombre. Aquestes llistes comencen amb l'etiqueta <ol> i després cada un dels elements de la llista es col·loquen en un <li>.

En les **llistes no numerades** cada un dels elements de la llista està precedit per un símbol, que per defecte és un cercle. Es comença la definició per <ul> i posteriorment els elements es posen dins d'un <li>.

Les **llistes de definició** són les menys conegudes però permeten especificar dos valors per cada element: un per a un terme o una frase, i l'altre per a la definició. En aquestes llistes es comença la llista per <dl>, els termes per <dt> i la definició per <dd>.

**FIGURA 1.50.** Representació de la informació en llistes

Llista numerada	Llista no numerada	Llista de definicions
1. XML 2. HTML 3. XHTML	<ul style="list-style-type: none"> <li>• XML</li> <li>• HTML</li> <li>• XHTML</li> </ul>	<b>XML</b> Extensible Markup Language <b>HTML</b> HyperText Markup Language <b>XHTML</b> Extensible HyperText Markup Language

## Taules

A vegades la presentació de dades requereix que aquestes siguin formatades en taules. L'HTML permet la definició de taules amb tot un grup d'etiquetes.

Les taules sempre es consideren un grup de files en què cada fila té unes quantes caselles. Hi ha un grup més d'etiquetes per representar capçaleres, peus de taula, unir caselles, però la idea sempre es basa en el mateix.

Les etiquetes bàsiques per representar les taules són tres: `<table>` per marcar l'inici d'una taula, `<tr>` per crear files, i la que representarà cada una de les cel·les dins d'una fila, `<td>`.

El codi següent ens crea una taula de dues files i dues caselles a cada fila:

```

1 <table>
2   <tr>
3     <td>1</td>
4     <td>2</td>
5   </tr>
6   <tr>
7     <td>3</td>
8     <td>4</td>
9   </tr>
10 </table>

```

En ser representat, aquest codi ens donarà un document com el que es mostra a la figura 1.51 (s'hi han afegit les línies per poder visualitzar-ne millor el resultat).

**FIGURA 1.51.** Representació d'una taula

1	2
3	4

## Esquema bàsic d'un document HTML

Els documents HTML només tenen una arrel, que serà l'element `<html>`. Aquesta etiqueta es fa servir per informar a qui llegeixi el document que el contingut del document és un fitxer HTML.

L'arrel només pot tenir dos elements fills:

- <head>: és el lloc en el qual es pot posar informació sobre el document HTML. El títol, el creador, l'idioma, el full d'estil, etc. Molta d'aquesta informació no es visualitzarà en el navegador.
  - En la capçalera hi ha un element obligatori, <title>, en el qual s'especificarà el nom del document.
- <body>: conté el contingut i l'estructura del document. Són les parts que es “veuen” en visualitzar el document HTML.

## Declaració DOCTYPE

Segons els estàndards HTML cada document necessita una declaració del tipus de document, que ha de sortir abans de l'element <html>. Encara que no és estrictament necessari, és molt recomanable.

Amb l'etiqueta DOCTYPE es defineix quin és el vocabulari real que es fa servir en el document i quin és el tipus d'estàndard que es fa servir, i es dóna informació al possible validador sobre quina versió ha de fer servir per comprovar la sintaxi del document.

Tot i que en les primeres versions no es feien servir, també s'han definit DOCTYPE per a aquestes versions:

- **HTML 2**

---

<sup>1</sup> `<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">`

---

- **HTML 3.2**

---

<sup>1</sup> `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">`

---

Per a la **versió 4.01** es van crear diferents conjunts de regles en funció d'una sèrie d'objectius. L'objectiu era que tothom fes servir la versió *strict*, però per criteris de compatibilitat es van definir altres versions.

- **Strict:** no permet informació presentacional ni els elements declarats “per eliminar”. Tampoc no permet fer servir marcs.

---

<sup>1</sup> `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"`  
<sup>2</sup> `"http://www.w3.org/TR/html4/strict.dtd">`

---

- **Transitional:** permet elements i atributs que ja no es recomanen. Està pensat per compatibilitat respecte a versions anteriors. No es permeten marcs.

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

- **Frameset:** és idèntic al *transitional* però accepta marcs

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
```

XHTML també ha definit els seus DTD i a més la seva especificació defineix que s'han d'especificar obligatòriament:

- **XHTML 1.0 strict**

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

- **XHTML 1.0 transitional**

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

- **XHTML 1.0 frameset**

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

En **XHTML 1.1** això ja no es va fer d'aquesta manera i només hi ha una sola definició DOCTYPE per a tots els documents. XHTML 1.1 és igual que la versió 1.0 *strict* però hi afegeix suport per a mòduls.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

## Esquelet bàsic

Per tant, si escollim fer servir *HTML 4.01 strict*, l'esquelet bàsic d'un document XHTML serà com aquest:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
3
4 <html>
5   <head>
6     <title>Titol</title>
7   </head>
8   <body>
9     ...
10  </body>
11 </html>
```

## Enllaços

La possibilitat que els documents s'enllacen entre ells ha estat un dels aspectes clau en la popularitat de les pàgines web. L'objectiu dels enllaços és permetre tenir un apuntador unidireccional cap a una destinació qualsevol i que s'hi pugui accedir per mitjà d'una URL.

Hi ha diverses maneres de fer enllaços en HTML però la més corrent és l'etiqueta `<a>`. L'objectiu d'aquesta etiqueta és definir origen i destinacions dels enllaços.

L'únic obligatori que té aquesta etiqueta és `href`, que permet definir la destinació.

Les destinacions solen ser altres documents HTML.

```
1 <a href="http://ioc.xtec.cat/">Web de l'I0C</a>
```

O documents d'altres tipus:

```
1 <a href="http://ioc.xtec.cat/educacio/presentacio.mp3">Presentació</a>
```

O fins i tot es poden enllaçar punts dins del document mateix. Per fer-ho només s'ha d'afegir el nom d'un origen que s'hi hagi definit i posar-hi el símbol # davant.

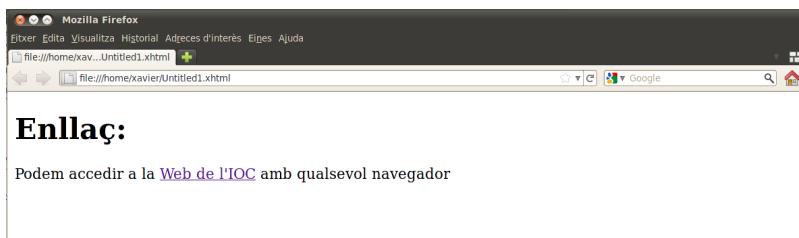
```
1 <a href="#LL0C">Enllaç dins del document</a>
```

Els punts que poden ser destinacions dins d'un document es defineixen amb la mateixa etiqueta però fent servir l'atribut 'id' i un nom descriptiu únic. Per exemple, si es col·loca aquest codi en un document estem definint que pugui ser enllaçat amb el nom LL0C.

```
1 <a id="LL0C" />
```

Tots els navegadors per defecte mostren els enllaços de manera diferent del text normal per facilitar que el lector els localitzi ràpidament (figura 1.52).

**FIGURA 1.52.** Els enllaços es mostren de manera diferent per ser localitzats fàcilment



## Formularis

Un formulari en HTML és una part especial que conté uns elements anomenats *controls* que permeten que l'usuari hi introdueixi algun tipus de dades.

Els formularis sobretot estan pensats per permetre que els usuaris enviiïn informació emplenant o seleccionant els controls que conté.

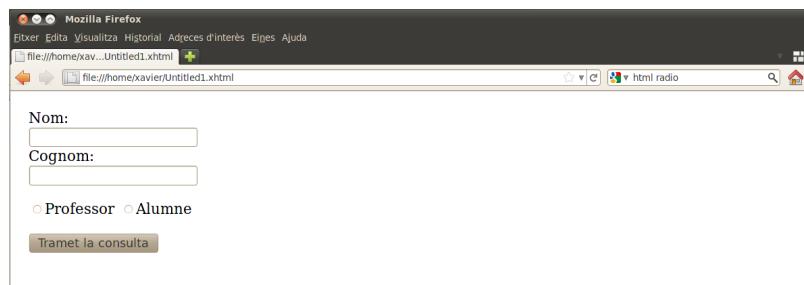
Així, el codi següent:

```

1  <form action="resultat.php" method="post">
2      <p>
3          <label for="nom">Nom:</label>
4          <input type="text" id="Nom" />
5          <label for="cognom">Cognom:</label>
6          <input type="text" id="Cognom" />
7          <input type="radio" name="tipus" value="professor"/>
8          <input type="radio" name="tipus" value="alumne"/>
9          <input type="submit">
10     </p>
11 </form>
```

...ens mostrarà el resultat que podeu veure en la figura 1.53.

**FIGURA 1.53.** Formulari en HTML



## 1.2.2 Convertir d'HTML a XHTML

XHTML (*extensible hypertext markup language*) és una recomanació oficial del W3C i defineix una sintaxi d'HTML compatible amb XML. O dit d'una altra manera, és un vocabulari XML vàlid.

Tot i que visualment l'HTML i l'XHTML són molt semblants, la diferència més important entre els dos llenguatges és que l'XHTML compleix les regles XML i per tant no permet que hi hagi etiquetes que s'obrin i no es tanquin o que no es mantingui l'ordre a l'hora de tancar les etiquetes obertes.

El compliment de les normes XML fa que els documents XHTML tinguin tendència a ser més grans però també tenen l'avantatge de que poden ser processats d'una forma més senzilla pels programes i que això els fa més fàcils de mantenir.

La segona diferència important està en el fet de que durant els anys l'evolució d'HTML ha fet que hi apareguin algunes inconsistències en les que XHTML ha intentat posar ordre. Per exemple HTML fa servir en alguns elements els atributs *id* i *name* per pràcticament les mateixes coses.

XHTML parteix de la idea actual de separar la presentació de les dades del seu contingut i per tant elimina algunes de les etiquetes d'HTML que es feien servir exclusivament per donar format (*font*, *center*, *u*, *s*, *strike*, *color*, *align*...). L'XHTML ja no es preocupa de l'aspecte de les coses sinó de què signifiquen.

La manera com s'ha de representar es deixa per als navegadors o bé per als fulls d'estil.

Però també hi ha algunes diferències més com que mentre en HTML l'especificació del DTD que es fa servir és opcional en XHTML s'ha d'especificar sempre o que XHTML obliga a fer servir les minúscules per especificar les etiquetes i els atributs

Com que l'HTML deixava molta llibertat, molts documents necessiten bastants canvis per poder-se convertir en documents XHTML. Però com a contraprestació, un cop fet el canvi tindran l'avantatge que els documents obtinguts seran documents XML vàlids, i per tant podran aprofitar dels avantatges que ofereix l'XML. Es podran processar amb les mateixes eines que els documents XML i n'obtindran els avantatges derivats:

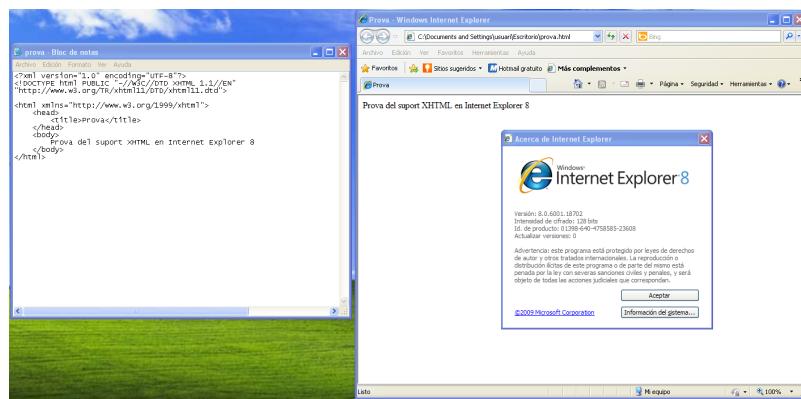
- Seran més fàcils d'usar pels programes d'ordinador.
- Es podran validar els documents amb els processadors XML.
- Seran més petits perquè no contindran la informació sobre el format per aplicar.
- Serà més senzill canviar-ne la presentació, ja que el codi HTML no s'haurà de tocar. I per tant, el mateix document es podrà representar de maneres alternatives i es podrà adaptar als nous dispositius.
- S'hi podrà mesclar informació en altres llenguatges XML (SVG, MathML, etc..).

## Declaració XML

La declaració XML és opcional en els documents XML però es recomana que tots els documents XHTML en tinguin.

<sup>1</sup> `<?xml version="1.0" encoding="UTF-8"?>`

Però a pesar de la recomanació la realitat és que no es fa servir gaire perquè un dels navegadors més usats, el Microsoft Internet Explorer, durant molt de temps, en trobar la declaració XML mostrava l'arbre XML en comptes de mostrar la pàgina web. Com es pot veure en la figura 1.54, en les darreres versions de l'Internet Explorer ja no és així, i per tant no hi ha excusa per no incloure la declaració XML en els documents HTML.

**FIGURA 1.54.** Internet Explorer 8 ja no té problemes amb la declaració XML

Per tant, podem trobar molts documents que són XHTML però que no tenen la declaració XML, tot i que es recomana fer-la servir.

vegeu [La declaració DOCTYPE](#)

## DOCTYPE

L'estàndard XHTML especifica que la declaració DOCTYPE és obligatòria. Per tant si es vol convertir un document HTML que no contingui la declaració DOCTYPE, en HTML era opcional, en un XHTML s'hi haurà d'afegir

### Espai de noms

Com que l'XHTML està basat en XML, s'ha de definir l'espai de noms d'XHTML en l'etiqueta arrel :

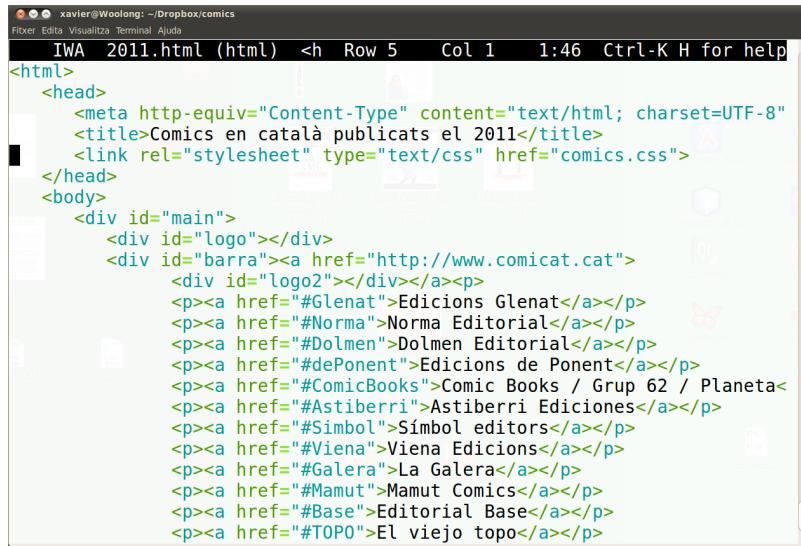
```
1  <html xmlns="http://www.w3.org/1999/xhtml">
2  ...
3  </html>
```

A més, tot i que no és obligatori, es recomana fer servir els atributs `xml:lang` i `lang` per definir l'idioma en el qual està escrit el document.

```
1  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ca" lang="ca" >
2  ...
3  </html>
```

### 1.2.3 Eines de disseny web

Els documents HTML/XHTML només són documents de text i, per tant, es poden crear fent servir un simple editor de text, com es pot veure en la imatge (figura 1.55).

**FIGURA 1.55.** Edició HTML des de consola


```

xavier@Woolong: ~/Dropbox/comics
Fitxer Edits Visualitzar Terminal Ajuda
IWA_2011.html (html) <h Row 5 Col 1 1:46 Ctrl-K H for help
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
    <title>Comics en català publicats el 2011</title>
    <link rel="stylesheet" type="text/css" href="comics.css">
  </head>
  <body>
    <div id="main">
      <div id="logo"></div>
      <div id="barra"><a href="http://www.comicat.cat">
        <div id="logo2"></div></a><p>
        <p><a href="#Glenat">Edicions Glenat</a></p>
        <p><a href="#Norma">Norma Editorial</a></p>
        <p><a href="#Dolmen">Dolmen Editorial</a></p>
        <p><a href="#dePonent">Edicions de Ponent</a></p>
        <p><a href="#ComicBooks">Comic Books / Grup 62 / Planeta</a></p>
        <p><a href="#Astiberri">Astiberri Ediciones</a></p>
        <p><a href="#Simbol">Símbol editors</a></p>
        <p><a href="#Viena">Viena Edicions</a></p>
        <p><a href="#Galera">La Galera</a></p>
        <p><a href="#Mamut">Mamut Comics</a></p>
        <p><a href="#Base">Editorial Base</a></p>
        <p><a href="#TOP0">El viejo topo</a></p>
      </div>
    </div>
  </body>
</html>

```

A pesar d'això hi ha tota una sèrie d'editors que ofereixen diverses ajudes per crear documents HTML de manera més ràpida i menys subjecta a errors.

Molts editors ofereixen alguna ajuda visual per a la creació de pàgines web. El més habitual és l'acoloriment de sintaxi HTML o fins i tot autocompletar durant l'edició (figura 1.56).

**FIGURA 1.56.** Edició d'HTML amb el Komodo


```

67 |   <p class="autor">Masashi Kishimoto</p>
68 | </td>
69 | </tr>
70 | <tr>
71 | <td class="descripcion">
72 |   En Killer Bee allibera tot el poder del seu bijū i llança un contraatac sobre en Sasuke i els seus! L'inconcebible chakra
73 |   que es desplega davant seu acròala en Sasuke i noqueja els seus companys! Què faran ara? D'altra banda, mentre en Naruto
74 |   s'esforça a aprendre les tècniques senjutsu, un perill sense precedents amenaça la vila de Konoha!
75 |
76 | </td>
77 | <td>
78 | <table border="1">
79 | <tr>
80 | <td>
81 | 
82 | </td>
83 | <td>46-Volum</td>
84 | <td>Masashi Kishimoto</td>
85 | </tr>
86 | <tr>
87 | <td class="descripcion">
88 |   Els sis Pain ataquen amb violència i transformen Konoha en un camp de batalla! Els ningües de la vila, tret de l'absent
89 |   Naruto, lluiten en bloc per enfocar-se a l'amença, però les tècniques desconegudes de l'enemic fan estralls entre les seves
90 |   files. En Kakashi, que ha degut tot el seu chakra, s'ha d'enfrontar ara a Tendō!
91 |
92 | </td>
93 | <td>
94 | 
95 | </td>
96 | <td>47-Volum</td>
97 | </tr>

```

A pesar de tot, els editors més populars són els editors HTML WYSIWYG (*what you see is what you get*) que permeten editar els documents HTML veient en tot moment com quin serà el resultat final.

## Editors HTML WYSIWYG

L'objectiu principal d'aquests editors és permetre que qualsevol pugui crear documents HTML per crear pàgines sense ser un expert en el llenguatge. S'esforcen a fer que es pugui treballar gràficament en comptes de fer-ho editant el codi HTML.

Hi ha una gran llista d'editors d'aquest tipus, i no para de créixer:

- Adobe Dreamweaver
- Microsoft Expression Studio

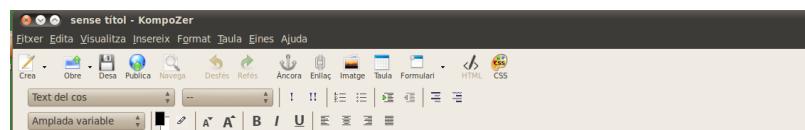
- CoffeCup HTML Editor
- Kompozer
- Amaya
- BlueFish
- Quanta Plus
- etc.

Sempre s'haurien de fer proves per determinar quin és l'editor HTML que s'adapta més a les necessitats que es tinguin.

### Barra d'eines

Normalment els editors basen el seu funcionament en una sèrie de botons en forma de barra d'eines (figura 1.57) que permeten arrossegar components sobre la pantalla d'edició.

**FIGURA 1.57.** Barra d'eines del Kompozer



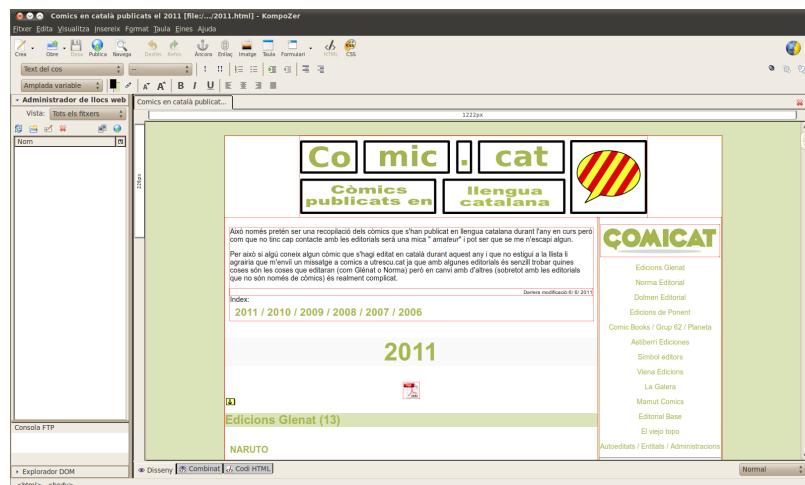
### Vistes

Els editors HTML solen dividir la pantalla en diferents vistes per intentar facilitar les tasques als usuaris. En cada una de les vistes de treball l'usuari pot veure diferents aspectes de la web en funció de les necessitats de cada moment.

Normalment sempre solen tenir almenys dues vistes, una **vista de previsualització** (figura 1.58) i una **vista de codi**.

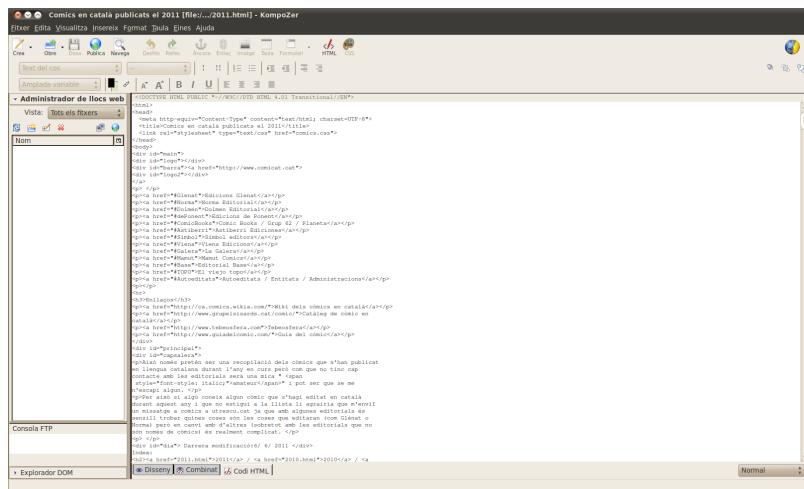
El més habitual serà treballar amb la **vista de previsualització**, ja que permet editar l'arxiu a més de veure en tot moment quin és l'aspecte que va prenent la pàgina.

**FIGURA 1.58.** Vista de previsualització del Kompozer



En la **vista de codi** (figura 1.59) es pot veure i editar el codi HTML que ha anat generant el programa automàticament. Normalment es recorre a aquesta vista quan cal fer alguna tasca que no està prevista en l'editor gràfic, o per forçar que el codi compleixi amb l'estàndard.

**FIGURA 1.59.** Vista de codi del KompoZer



## Característiques extra

La major part dels editors solen oferir la possibilitat de fer més tasques a part de l'edició d'HTML assistida:

- Permetre l'edició de fulls d'estil.
- Facilitar la incorporació de codi en llenguatges de programació a la pàgina: Javascript, PHP...
- Penjar la pàgina al servidor on ha de ser-hi.
- etc.

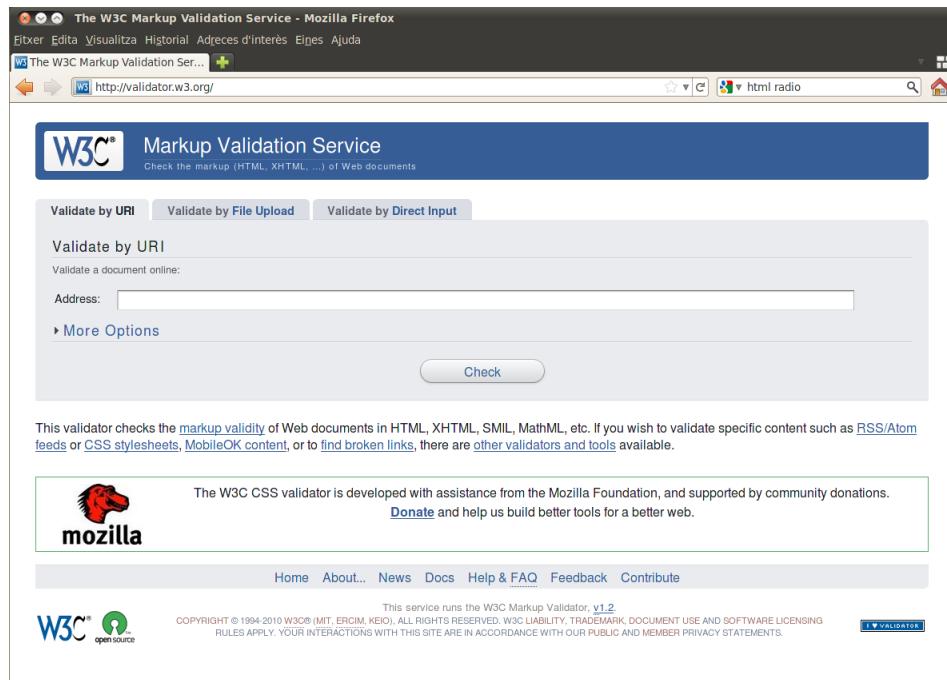
## Problemes

Però a pesar dels avantatges que ofereixen aquests editors també s'hi poden trobar alguns punts foscos:

- No sempre són gaire estrictes en el seguiment dels estàndards.
- No sempre tots els navegadors interpreten igual que els editors les pàgines i, per tant, podem tenir resultats inesperats.

Per tant, sempre que es dissenyi una pàgina s'hauria de validar el codi amb un validador i a més comprovar quin és el resultat de visualitzar la pàgina amb diferents navegadors.

Un dels validadors en línia més populars és el que ofereix el W3C a <http://validator.w3.org> (figura 1.60).

**FIGURA 1.60.** Validador HTML/XHTML del W3C

Aquest validador a partir de la línia DOCTYPE analitza el document que hem fet i ens mostra els errors que hi trobi.

També ens permet comprovar quin seria el resultat si definissim algun altre DOCTYPE. Per exemple, si validem la web de l'IOC contra XHTML 1.1 ens dóna 8 errors i ens explica com solucionar-los (figura 1.61).

**FIGURA 1.61.** Errors en validar la web de l'IOC contra XHTML 1.1

Errors found while checking this document as XHTML 1.1!	
<b>Result:</b>	8 Errors, 1 warning(s)
<b>Address:</b>	<input type="text" value="http://ioc.xtec.cat/educacio/"/>
<b>Encoding:</b>	utf-8 ( <input type="button" value="detect automatically"/> )
<b>Doctype:</b>	XHTML 1.1 ( <input type="button" value="XHTML 1.1"/> )
<b>Root Element:</b>	html
<b>Root Namespace:</b>	<a href="http://www.w3.org/1999/xhtml">http://www.w3.org/1999/xhtml</a>

**Validation Output: 8 Errors**

- ✖ Line 37, Column 36: document type does not allow element "label" here; missing one of "ins", "del", "h1", "h2", "h3", "h4", "h5", "h6", "p", "div", "address", "fieldset" start-tag  
`<label for="mod_search_searchword">`  
The mentioned element is not allowed to appear in the context in which you've placed it; the other mentioned elements are the only ones that are both allowed there and can contain the element mentioned. This might mean that you need a containing element, or possibly that you've forgotten to close a previous element.  
One possible cause for this message is that you have attempted to put a block-level element (such as "<p>" or "<table>") inside an inline element (such as "<a>", "<span>", or "<font>").
- ✖ Line 39, Column 230: document type does not allow element "input" here; missing one of "ins", "del", "h1", "h2", "h3", "h4", "h5", "h6", "p", "div", "address", "fieldset" start-tag  
`...s.value=='cerca...') this.value=''; /><input type="submit" value="Cerca" class="`  
The mentioned element is not allowed to appear in the context in which you've placed it; the other mentioned elements are the only ones that are both allowed there and can contain the element mentioned. This might mean that you need a containing element, or possibly that you've forgotten to close a previous element.  
One possible cause for this message is that you have attempted to put a block-level element (such as "<p>" or "<table>") inside an inline element (such as "<a>", "<span>", or "<font>").

## 2. Definició d'esquemes i vocabularis en XML

L'XML permet crear els llenguatges de marques que vulguem, sigui quin sigui el camp d'actuació, ja que en deixar crear les etiquetes que calguin es pot adaptar a qualsevol situació. Aquest és el punt fort de l'XML sobre altres llenguatges de marques: s'adapta al que es vulgui representar sense importar la complexitat que pugui tenir.

Però les dades dels documents XML normalment hauran de ser processades per un programa d'ordinador, i els programes d'ordinador no donen tanta llibertat com l'XML. Aquests no són gaire bons interpretant i entenent informació per a la qual no han estat programats per processar.

Suposem que s'ha desenvolupat un programa per representar imatges a partir de les etiquetes `<dibuix>`, `<rectangle>`, `<cerclle>`. Des d'un punt de vista de la llibertat que deixa l'XML no hi haurà cap problema per crear un arxiu XML com aquest:

```

1 <dibuix>
2   <rectangle>12,10,14,10</rectangle>
3   <linia>13,13,25,25</linia>
4 </dibuix>
```

El document és perfectament correcte des d'un punt de vista de l'XML però el programa no sabrà què fer amb l'etiqueta `<linia>` perquè ningú no l'ha programat. És per aquest motiu que els programes normalment es dissenyen per processar només tipus concrets d'XML.

Per tant, una de les coses que haurà de fer un programa és comprovar que les dades del document XML siguin correctes. Com que aquesta tasca és molt feixuga s'han definit sistemes per comprovar que el document XML entrat conté les etiquetes que ha de contenir i que estan col·locades tal com fa falta. El procés de fer aquestes comprovacions s'anomena **validació**.

### 2.1 Validació de documents XML

El procés de comprovar que uns fitxers XML determinats segueixen un determinat vocabulari s'anomena **validació** i els documents XML que segueixen les regles del vocabulari s'anomenen **documents vàlids**. Cal tenir clara la diferència entre el que és un document ben format i un document vàlid:

Un document és **ben format** quan segueix les regles d'XML.

Un document és **vàlid** si segueix les normes del vocabulari que té associat.

Un document pot complir perfectament amb les regles de creació de documents XML i, per tant, estar **ben format**, però en canvi no seguir les normes del vocabulari i, per tant, **no ser vàlid**.

Un document pot ser ben format però no vàlid, i en canvi si és vàlid segur que és ben format.

La validació de documents és un procés corrent en llenguatges de marques, ja que no sempre la persona que defineix com ha de ser el document després és la que generi els documents (de vegades els documents arribaran d'altres llocs, es generaran automàticament...).

La validació és el procés de comprovar que un document compleix amb les regles del vocabulari en tots els seus aspectes.

La validació permet assegurar que la informació està exactament en la manera com ha d'estar. Això és especialment important quan es comparteix informació entre sistemes, ja que validar el document és assegurar-se que realment l'estructura de la informació que s'està passant és la correcta. Si es vol fer que dos programes situats en ordinadors diferents col·laborin cal que la informació que es passen l'un a l'altre segueixi l'estructura prefixada per enviar-se missatges.

Per forçar una determinada estructura en un document cal que hi hagi alguna manera de definir aspectes com ara:

- en quin ordre han d'anar les etiquetes,
- quines són correctes i quines no,
- en quin ordre han d'aparèixer,
- quins atributs s'hi poden posar,
- quin contingut hi pot haver,
- etc.

L'XML fa això per mitjà de **llenguatges de definició de vocabularis** o **llenguatges d'esquemes**.

Els llenguatges de definició d'esquemes sorgeixen per la necessitat que els documents XML siguin processats per programes (per extreure'n informació, transformar-los en altres coses, etc), ja que els programes no són tan flexibles com l'XML i necessiten processar dades amb estructures de document tancades.

Hi ha molts llenguatges de definició d'esquemes, però els més utilitzats en XML són:

- *Document type definitions* (DTD)

- W3C XML definition language
- Relax NG
- Schematron

El procés de validació es fa per mitjà de programes especials anomenats **processadors o validadors** (en anglès s'anomenen **parsers**).

Els validadors sovint estan en forma de biblioteques per poder ser incorporats als programes. No tots els processadors poden validar tots els llenguatges de definició de vocabularis i, per tant, s'hauran de triar en funció del llenguatge que fem servir.

## 2.2 Processadors d'XML

Els processadors XML són els encarregats de validar els documents i, per tant, és molt important triar un processador que sigui capaç de validar el llenguatge de validació que fem servir:

- El fet que la DTD sigui el sistema més antic i que se'n parli en l'especificació d'XML ha fet que la majoria dels processadors XML puguin validar llenguatges definits amb DTD.
- Amb el pas del temps, XML Schemas s'ha convertit en la manera estàndard de validar vocabularis XML, i altres tecnologies XML en fan servir alguns aspectes. Per tant, la majoria dels processadors també suporten aquest format.
- Molts processadors a part dels dos més usats també poden fer servir altres llenguatges (en especial Relax NG).

### 2.2.1 Biblioteques per validar XML

Els documents XML estan pensats per poder ser tractats de manera automàtica, i moltes de les utilitats per validar documents estan en forma de biblioteques. Entre les biblioteques existents per validar XML en podem destacar:

- libxml2
- MSXML
- Apache Xerces2

## libxml2

La utilitat *libxml* de la biblioteca *libxml2* del projecte Gnome permet validar documents XML que tinguin associats DTD, XML Schemas, Relax NG o Schematron.

### Validar DTD

La manera més senzilla de fer-ho és especificar el paràmetre **-valid**, que fa que el programa, a més de comprovar que el document XML estigui ben format, sigui vàlid.

Amb el paràmetre **-noout** no es mostrarà cap missatge si el document és correcte.

```
1 $ xmllint --valid prova.xml --noout
2 $
```

També ofereix l'opció de validar un document extern amb el paràmetre **-dtdvalid**. Això és especialment interessant per poder validar sense haver d'associar el document XML i la DTD.

```
1 $ xmllint alumnes.xml --dtdvalid alumnes.dtd --noout
2 $
```

En cas que el document no validi mostrarà l'error trobat:

```
1 $ xmllint --valid prova.xml --noout
2 prova.xml:13: element classe: validity error :
3 Element classe content does not follow the DTD,
4 expecting (classe , professor , alumnes), got (professor alumnes )
5 </classe>
6 ^
```

### Validar XML Schemas

Per validar un document amb un XML Schema s'ha d'especificar el document d'esquema fent servir el paràmetre **-schema**:

```
1 xmllint --schema classe.xsd classe.xml --noout
2 classe.xml validates
```

Si no valida mostrarà els errors que hi hagi detectat:

```
1 $ xmllint --schema classe.xsd classe.xml --noout --valid
2 classe.xml:3: element curs: Schema's validity error :
3 Element 'curs': This element is not expected. Expected is ( professor ).
4 classe.xml fails to validate
```

### Validar amb Relax NG i Schematron

No hi ha massa diferències a l'hora de validar contra Relax NG o Schematron. Simplement s'especifica el paràmetre correcte. Per a Relax NG és **-relaxng**:

```
1 $ xmllint --relaxng classe.rng classe.xml --noout
2 classe.xml validates
```

## MSXML

Des dels entorns Windows es pot fer servir la biblioteca MSXML per validar els documents XML contra DTD o XML Schema.

Aquesta biblioteca ofereix una interfície per desenvolupar aplicacions que treballin amb diferents tecnologies XML. Es fa servir en molts dels programes de Microsoft i és la que fan servir per defecte els programes basats en la plataforma .NET (C#, Visual Basic, J#, managed C++ ...).

Alguns dels editors especialitzats en XML que funcionen exclusivament en Windows fan servir aquesta biblioteca per funcionar.

## Apache Xerces2

Aquesta biblioteca està pensada per ser usada des de programes i no des de la línia d'ordres (per exemple, molts dels editors especialitzats en XML fan o permeten fer servir Xerces2 per validar els documents XML). A pesar d'això, sí que és possible validar un document des de la línia d'ordres si s'instal·len els exemples, ja que contenen un programa anomenat **Counter** que permet fer estadístiques d'un document i que alhora valida.

En una instal·lació d'Ubuntu s'executaría el programa i donaria el resultat següent:

```

1 $ java -cp /usr/share/java/xercesImpl.jar:\ 
2 >/usr/share/java/xmlParserAPIs.jar:\ 
3 >/usr/share/doc/libxerces2-java-doc/examples/xercesSamples.jar \ 
4 dom/Counter -v alumnes.xml
5 [Error] alumnes.xml:7:11: The content of element type "alumne" is incomplete,
       it must match "(nom,cognom,cognom)".
6 [Error] alumnes.xml:11:11: The content of element type "alumne" is incomplete,
       it must match "(nom,cognom,cognom)".
7 [Error] alumnes.xml:15:11: The content of element type "alumne" is incomplete,
       it must match "(nom,cognom,cognom)".
8 alumnes.xml: 229;15;0 ms (10 elems, 9 attrs, 31 spaces, 36 chars)
```

Si el document valida correctament mostrarà estadístiques del document:

```

1 $ java -cp /usr/share/java/xercesImpl.jar:\ 
2 >/usr/share/java/xmlParserAPIs.jar:\ 
3 >/usr/share/doc/libxerces2-java-doc/examples/xercesSamples.jar \ 
4 >dom/Counter -v alumnes.xml
5 alumnes.xml: 39;7;0 ms (10 elems, 9 attrs, 31 spaces, 36 chars)
```

En estar desenvolupat en Java també pot ser executat en entorns Windows. Suposant que tinguem Xerces a C:\xerces es pot validar de la mateixa manera:

```

1 c:\> java -cp c:\xerces\xercesImpl.jar;
2 c:\xerces\xmlParserAPIs.jar;
3 c:\xerces\xercesSamples.jar dom/Counter -v alumnes.xml
4 alumnes.xml: 39;7;0 ms (10 elems, 9 attrs, 31 spaces, 36 chars)
```

## 2.2.2 Programes

Tot i que els validadors sobretot es fan servir com a biblioteques incorporades en programes, també hi ha programes que permeten validar documents sense haver d'escriure ni una línia de codi. Exemples d'aquests tipus de programes poden ser XMLStarlet o Jing.

### Validar amb XMLStarlet

L'XMLStarlet és un programa de codi obert que s'executa des de línia d'ordres i que pot manipular fitxers XML. Fa servir *libxml2* i es pot baixar des de <http://xmlstar.sourceforge.net/>.

Una de les tasques que s'hi poden fer és validar documents XML definits amb DTD, XML Schemas i Relax NG. Si el document ja té associada la DTD n'hi ha prou d'executar la instrucció amb `validate` i el nom del fitxer:

```
1 $ xmlstarlet validate prova.xml  
2 prova.xml - valid
```

O es pot especificar un fitxer de definició extern:

```
1 $ xmlstarlet validate --dtd prova.dtd prova.xml  
2 prova.xml - valid
```

Es pot fer el mateix per validar XML Schemas:

```
1 $ xmlstarlet validate --xsd prova.xsd prova.xml  
2 prova.xml - valid
```

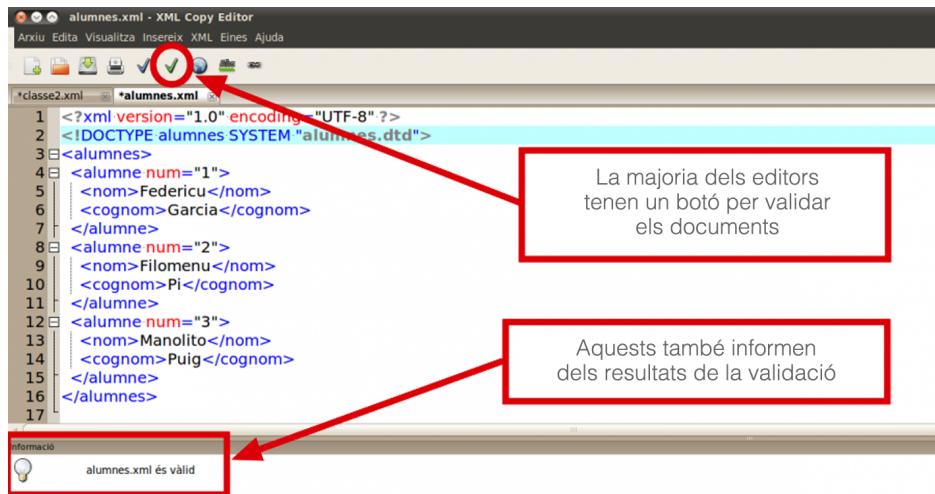
El problema que té el programa és que no informa dels errors trobats:

```
1 $ xmlstarlet validate --xsd classe.xsd classe.xml  
2 classe.xml - invalid
```

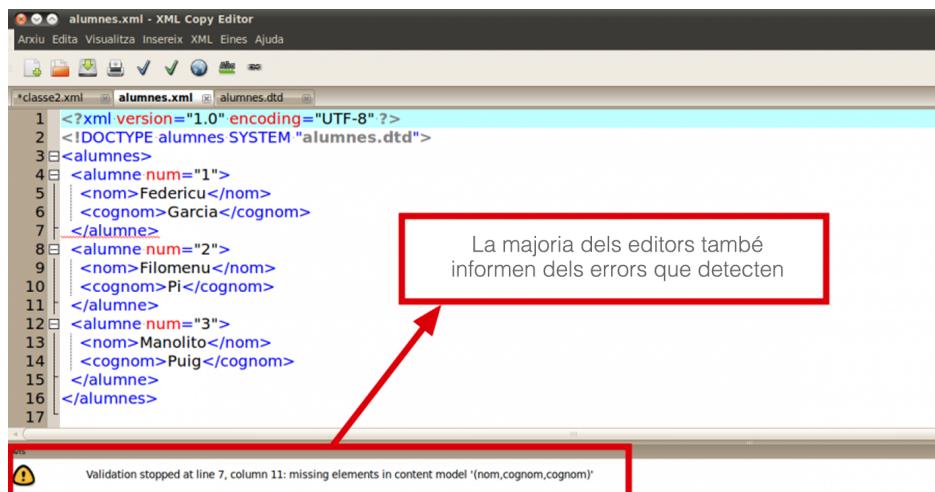
Hi ha molts programes que permeten validar documents XML sense haver de programar, però el més habitual des d'un punt de vista professional sol ser fer servir algun editor XML especialitzat, com ara oXygen XML Editor, Editix XML Editor, Altova XMLSpy XML editor o Stylus Studio.

## Editors XML

Els editors XML, a més de permetre la creació assistida de documents XML, també tenen alguna opció per fer validations dels documents sense haver d'abandonar l'editor (figura 2.1).

**FIGURA 2.1.** Validació des de l'XML Copy Editor

Aquests editors també detecten els problemes de validació automàticament i marquen les línies amb errors (figura 2.2).

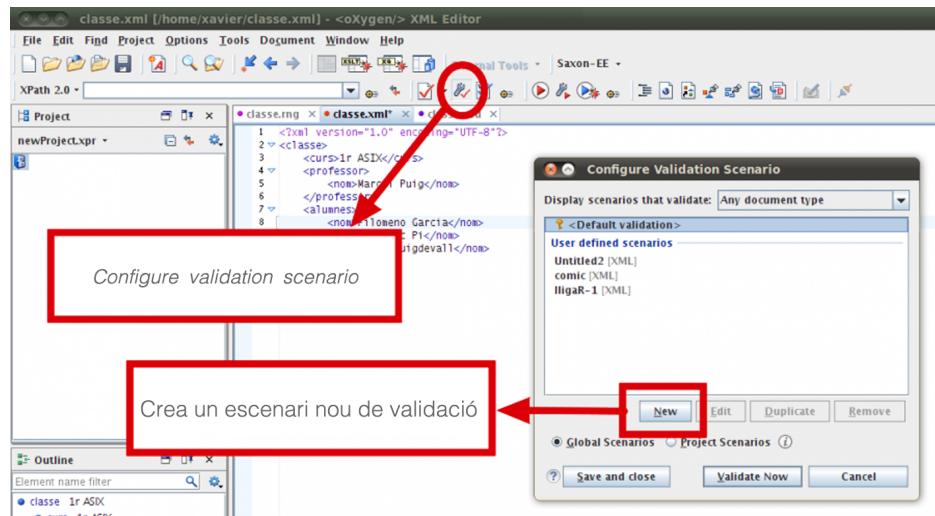
**FIGURA 2.2.** No cal sortir de l'entorn per veure'n els errors

Sovint també permeten validar documents XML contra gairebé tots els sistemes de validació: DTD, XML Schemas, Relax NG...

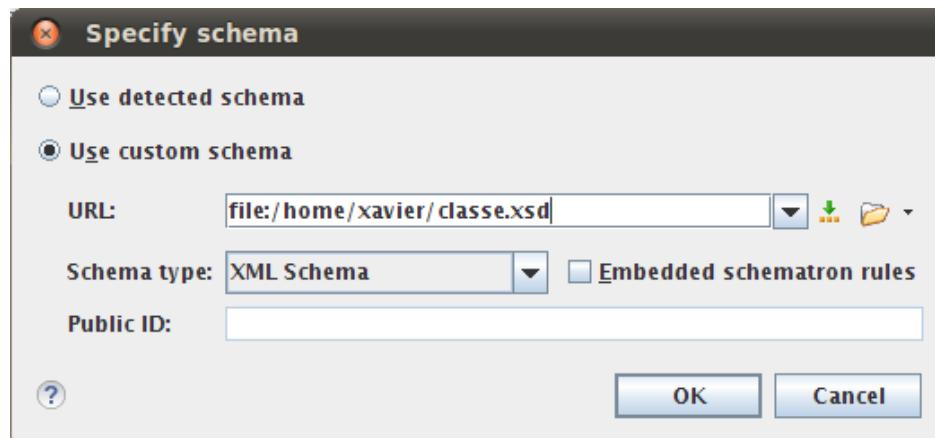
### Validar amb l'oXygen XML Editor

Com a exemple de validació amb un editor especialitzat XML farem una validació des de l'oXygen XML Editor.

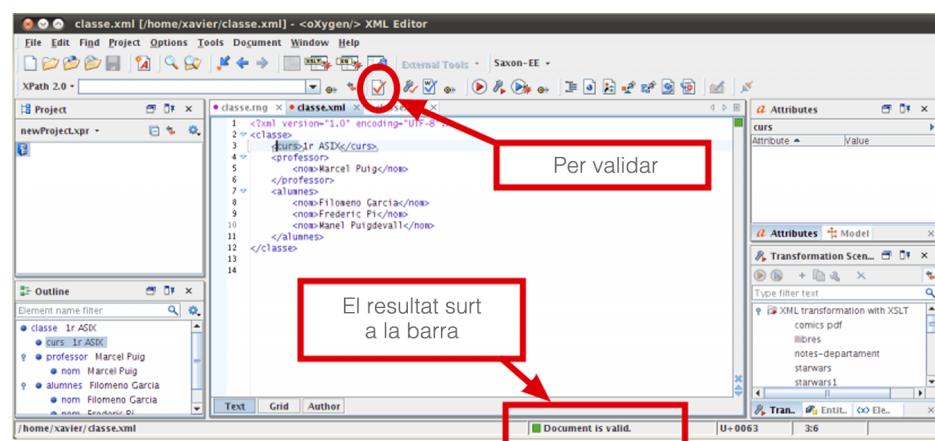
Un cop el document XML està carregat en el programa cal especificar contra quin document s'ha de validar. Això es fa des de l'opció *Configure Validation Scenario* (figura 2.3).

**FIGURA 2.3.** Configurar un escenari de validació amb l'oXygen

En la pantalla següent es defineix una nova validació i es tria el fitxer d'esquema (figura 2.4).

**FIGURA 2.4.** Tria del fitxer per fer la validació

El fitxer quedarà associat al document de manera que cada vegada que es validi el document ho farà contra l'esquema triat automàticament (figura 2.5).

**FIGURA 2.5.** Validació d'XML amb l'oXygen XML Editor 12

## 2.3 DTD

El DTD (*document type definitions*) és un llenguatge de definició d'esquemes que ja existia abans de l'aparició d'XML (es feia servir en SGML). En estar pensat per funcionar amb SGML es podia fer servir en molts dels llenguatges de marques que s'hi han basat, com XML o HTML.

Quan es va definir l'XML es va aprofitar per fer una versió simplificada de DTD que en fos el llenguatge d'especificació d'esquemes original. Un avantatge associat era que fer servir una versió de DTD permetria mantenir la compatibilitat amb SGML, i per tant es van referenciar les DTD en l'especificació d'XML (<http://www.w3.org/TR/REC-xml/>).

L'objectiu principal de les DTD és proveir un mecanisme per validar les estructures dels documents XML i determinar si el document és **vàlid** o no. Però aquest no serà l'únic avantatge que ens aportaran les DTD, sinó que també els podrem fer servir per compartir informació entre organitzacions, ja que si algú altre té la nostra DTD ens pot enviar informació en el nostre format i amb el programa que hem fet la podrem processar.

Durant molt de temps les DTD van ser el sistema de definir vocabularis més usat en XML però actualment han estat superats per XML Schemas. Tot i això encara és molt usat, sobretot perquè és molt més senzill.

### 2.3.1 Associar una DTD a un document XML

Per poder validar un document XML cal especificar-li quin és el document d'esquemes que es farà servir. Si el llenguatge que es fa servir és DTD hi ha diverses maneres d'especificar-ho però en general sempre implicarà afegir una declaració DOCTYPE dins el document XML per validar.

El més habitual és afegir la referència a la DTD dins del document XML per mitjà de l'etiqueta especial `<!DOCTYPE ...>`. Com que la declaració XML és opcional però si n'hi ha ha de ser la primera cosa que aparegui en un document XML, l'etiqueta DOCTYPE haurà d'anar sempre darrere seu.

Per tant, seria incorrecte fer:

---

```

1  <!DOCTYPE ... >
2  <?xml version="1.0"?>

```

---

Però, per altra banda, l'etiqueta no pot aparèixer en cap altre lloc que no sigui just a continuació de la declaració XML, i per tant tampoc no es pot incloure l'etiqueta DOCTYPE un cop hagi començat el document.

---

```

1  <?xml version="1.0"?>
2  <document>

```

---

```

3  <!DOCTYPE ...>
4  </document>
```

Ni tampoc al final:

```

1  <?xml version="1.0" ?>
2  <document>
3  </document>
4  <!DOCTYPE ... >
```

L'etiqueta DOCTYPE sempre ha d'anar o bé en la primera línia si no hi ha declaració XML, o bé just al darrere:

```

1  <?xml version="1.0" ?>
2  <!DOCTYPE ... >
3  <document>
4  </document>
```

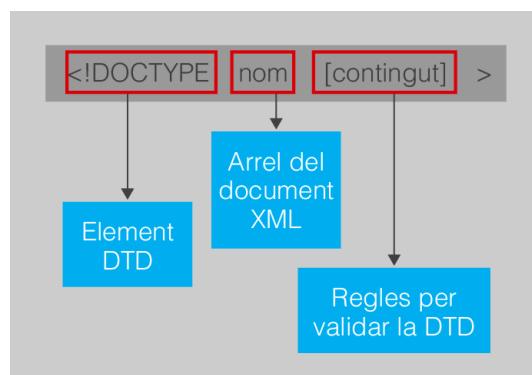
Hi ha dues maneres d'incorporar DTD en un document XML:

- Declaració interna
- Declaració externa

### Declaració DTD interna

En les declaracions DTD internes les regles de la DTD estan incorporades en el document XML. L'etiqueta DOCTYPE es declara com es veu en la figura 2.6.

**FIGURA 2.6.** Declaració DOCTYPE interna



Per exemple, si es copien les línies següents en un document XML anomenat prova.xml:

```

1  <?xml version="1.0"?>
2  <!DOCTYPE classe [
3      <!ELEMENT classe (professor, alumnes)>
4      <!ELEMENT professor (#PCDATA)>
5      <!ELEMENT alumnes (nom*)>
6      <!ELEMENT nom (#PCDATA)>
7  ]>
8  <classe>
9      <professor>Marcel Puig</professor>
10     <alumnes>
```

```
11 <nom>Fredderic Pi</nom>
12 </alumnes>
13 </classe>
```

Es pot comprovar que el document es valida fent servir la instrucció següent:

```
1 $ xmllint --valid prova.xml --noout
```

Les declaracions de DTD internes no es fan servir gaire perquè tenen una sèrie de problemes que no les fan ideals:

- En tenir la definició de l'esquema dins del document XML no és fàcil compartir les declaracions amb altres documents.
- Si es tenen molts documents XML, per canviar lleugerament la declaració s'hi hauran de fer canvis en tots.

És per aquest motiu que és més recomanable tenir la DTD en un document a part i fer servir aquest document DTD per validar tots els XML.

### Declaració DTD externa

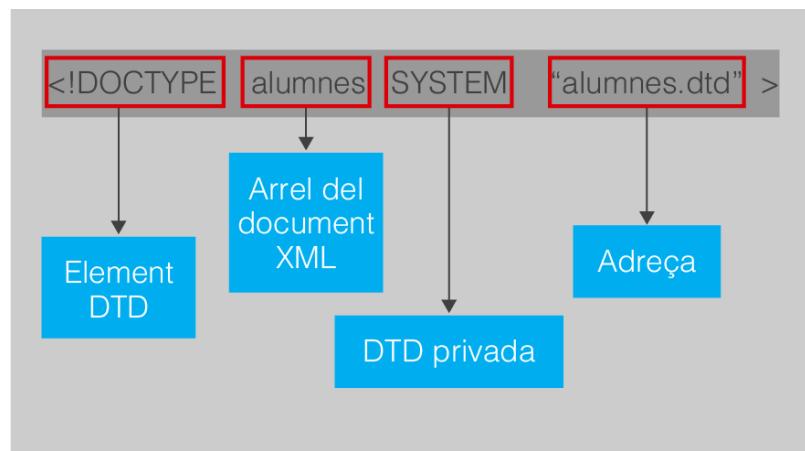
En comptes d'especificar la DTD en el mateix document es considera una pràctica molt millor definir-la en un fitxer a part.

Per fer una declaració externa també es fa servir l'etiqueta DOCTYPE però el format és lleugerament diferent i preveu dues possibilitats:

- DTD privada
- DTD pública

### DTD privades

Les definicions de DTD privades són les més corrents, ja que, tot i que es defineixi el vocabulari com a privat, no hi ha res que impedeixi que el fitxer es comarteixi o es publiqui per mitjà d'Internet. La definició d'una DTD privada es fa de la manera que es pot veure en la figura 2.7.

**FIGURA 2.7.** Definició d'una DTD privada en un document XML

El camp d'**adreça** especifica on és i quin nom té el fitxer que conté les regles. Es pot especificar fent servir una URI (*uniform resource identifier*). Això fa que es pugui definir la DTD per mitjà d'un nom que per definició hauria de ser únic.

En el camp d'adreça es poden definir camins dins la màquina:

**URI**

Una URI (*uniform resource identifier*) és una cadena de caràcters que es fa servir 'er fer referència única a un recurs local, dins d'una xarxa o a Internet.

```
1 <!DOCTYPE classe SYSTEM "C:\dtd\classe.dtd">
```

O fins i tot es pot definir una DTD per mitjà d'una adreça d'Internet:

```
1 <!DOCTYPE classe SYSTEM "http://www.ioc.cat/classe.dtd">
```

Per tant, podríem definir dins d'un fitxer XML una DTD que tingui l'element arrel `<alumnes>` d'aquesta manera:

```
1 <!DOCTYPE alumnes SYSTEM "alumnes.dtd">
```

Un cop definit només cal crear l'arxiu `alumnes.dtd` en el lloc adequat amb les regles que defineixen el vocabulari:

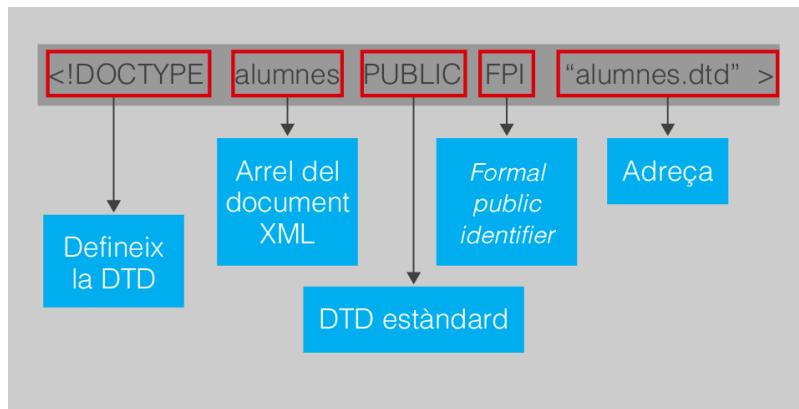
```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!ELEMENT alumne (professor, alumnes) >
3 <!ELEMENT professor (#PCDATA) >
4 <!ELEMENT alumnes (nom*) >
5 <!ELEMENT nom (#PCDATA) >

```

**DTD públiques**

Les definicions PUBLIC estan reservades per a DTD que estiguin definides per organismes d'estandardització, ja siguin oficials o no. En la definició d'una DTD pública s'afegeix un camp extra que fa d'identificador de l'organisme (figura 2.8).

**FIGURA 2.8.** Definició d'una DTD pública en un document XML

La majoria dels camps són idèntics que en definir una DTD privada però en aquest cas s'hi ha afegit un camp més, que és un identificador. L'identificador pot estar en qualsevol format però el més corrent és fer servir el format FPI (*formal public identifiers*). L'FPI es defineix per mitjà d'un grup de quatre cadenes de caràcters separades pels dobles barres. En cada posició s'especifica:

<sup>1</sup> "simbol//Nom del responsable de la DTD//Document descrit//Idioma"

en què cada valor es definirà segons la taula 2.1.

**TAULA 2.1.** Definició d'un FPI

Camp	Valors possibles
primer	Si l'estàndard no ha estat aprovat hi haurà un '-', mentre que si ha estat aprovat per un organisme no estàndard tindrem un '+'. I en canvi, si ha estat aprovat per un organisme d'estàndards, hi haurà una referència.
Nom del responsable	Qui és l'organització o la persona responsable de definir l'estàndard.
Document descrit	Conté un identificador únic del document descrit.
Idioma	El codi ISO de l'idioma en què està escrit el document.

Per exemple, aquesta seria una declaració correcta:

<sup>1</sup> <!DOCTYPE classe PUBLIC "-//IOC//Classe 1.0//CA" "http://www.ioc.cat/classe.dtd">

Un exemple de DTD pública que es fa servir molt és la declaració d'HTML 4.01:

<sup>1</sup> <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
<sup>2</sup> "http://www.w3.org/TR/html4/strict.dtd">

### Regles internes en declaracions externes

Qualsevol de les definicions externes també pot contenir un apartat opcional que permet especificar-hi un subconjunt de regles internes.

<sup>1</sup> <!DOCTYPE nom SYSTEM "fitxer.dtd" [ regles ] >

2    `<!DOCTYPE classe PUBLIC "-//ILOC//Classe 1.0//CA" "fitxer.dtd" [ regles ] >`

Si no hi ha regles no cal especificar aquest apartat però també es pot deixar en blanc. Per tant, seria correcte definir la DTD d'aquesta manera:

1    `<!DOCTYPE alumnes SYSTEM "alumnes.dtd" [ ]>`

Com d'aquesta:

1    `<!DOCTYPE alumnes SYSTEM "alumnes.dtd">`

### 2.3.2 Definició d'esquemes amb DTD

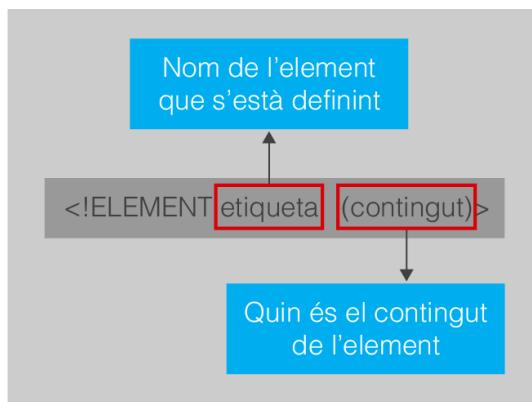
La part més important d'un sistema de definició de vocabularis és definir com es fa per determinar l'ordre en què els elements poden aparèixer i quin contingut poden tenir, quins atributs poden tenir les etiquetes, etc.

Això, en DTD, es fa definint una sèrie de regles per mitjà d'un sistema d'etiquetes predefinit. A diferència del que passa en XML, en fer una definició en DTD les etiquetes estan definides. Per definir elements i atributs s'hauran de fer servir etiquetes concretes.

#### Elements

De la mateixa manera que els elements són la base de l'XML, la definició d'aquests elements és la base de la definició de fitxers DTD. Per tant, sempre s'hauran de definir tots els elements que componen el vocabulari i a més especificar-ne el contingut (figura 2.9).

**FIGURA 2.9.** Definició d'una DTD



Per tant, és molt senzill definir elements amb DTD: simplement es defineix el nom de l'element i quin és el contingut.

1    `<!ELEMENT nom (#PCDATA)>`

Com en XML, els continguts poden ser dades o bé altres elements. Podem definir tres grans grups de maneres de definir els continguts en una DTD:

- Continguts genèrics
- Contingut d'elements
- Contingut barrejat

### Continguts genèrics

Hi ha tres continguts genèrics que es poden fer servir per definir elements: ANY, EMPTY i #PCDATA (vegeu la taula 2.2).

**TAULA 2.2.** Continguts genèrics en una DTD

Valor	Significat
ANY	El contingut de l'element pot ser qualsevol cosa.
EMPTY	L'element no té contingut.
#PCDATA	El contingut de l'etiqueta poden ser dades.

Els elements que estiguin definits amb ANY poden contenir qualsevol cosa dins seu. Tant etiquetes, com dades, o fins i tot una barreja de les dues coses.

Si es defineix persona d'aquesta manera:

```
1 <!ELEMENT persona ANY>
```

validarà tant elements que continguin dades:

```
1 <persona>Frederic Pi</persona>
```

com elements que continguin altres elements:

```
1 <persona>
2   <nom>Frederic</nom>
3   <cognom>Pi</cognom>
4 </persona>
```

Per tant, el contingut definit amb ANY és molt potent però també fa que es perdi el control de les coses que es poden validar amb la DTD, ja que ho validarà pràcticament tot.

Els elements en XML a vegades no cal que tinguin cap valor per aportar alguna informació, ja que el nom de l'etiqueta pot tenir algun tipus de significat semàntic.

L'etiqueta *<professor>* de la línia 4 de l'exemple següent no necessita tenir cap contingut perquè està implícit que la persona a la qual s'assigni l'etiqueta serà un professor.

```
1 </persones>
2   <persona>
3     <nom>Pere Martí</nom>
4     <professor/>
```

```

5   </persona>
6   <persona>
7       <nom>Marcel Peris</nom>
8   </persona>
9 </persones>
```

Els continguts definits amb EMPTY estan pensats per a les etiquetes que no tindran cap contingut dins seu. Per tant, l'element <professor> de l'exemple anterior es definiria d'aquesta manera:

```
1  <!ELEMENT professor EMPTY>
```

Aquesta definició serveix tant per als elements que es defineixen fent servir el sistema d'una sola etiqueta...

```
1  <professor/>
```

com per als que defineixen les etiquetes d'obertura i tancament.

```
1  <professor></professor>
```

El contingut genèric #PCDATA (*parser character data*) segurament és el més usat per marcar que una etiqueta només té dades en el seu contingut.

Si partim de l'exemple següent:

```

1  <persona>
2      <nom>Marcel</nom>
3      <cognom>Puigdevall</cognom>
4  </persona>
```

Es pot fer servir #PCDATA per definir el contingut dels elements <nom> i <cognom> perquè dins seu **només tenen dades**.

```

1  <!ELEMENT nom (#PCDATA)>
2  <!ELEMENT cognom (#PCDATA)>
```

Però no es pot fer servir, en canvi, per definir l'element <persona>, perquè dins seu no hi té només dades sinó que hi té els elements <nom> i <cognom>.

## Contingut d'elements

Una de les situacions habituals en un document XML és que un element dins del seu contingut en tingui d'altres.

Tenim diverses possibilitats per definir el contingut d'elements dins d'una DTD:

- Seqüències d'elements
- Alternatives d'elements
- Modificadors

Si mirem l'exemple següent veurem que tenim un element <persona> que conté dos elements, <nom> i <cognom>.

```

1 <persona>
2   <nom>Pere</nom>
3   <cognom>Martinez</cognom>
4 </persona>
```

Per tant, l'element <persona> és un element que té com a contingut una **seqüència d'altres elements**. En una DTD es defineix aquesta situació definint explícitament quins són els fills de l'element, separant-los per comes.

```
1 <!ELEMENT persona (nom,cognom)>
```

Mai no s'ha d'oblidar que les seqüències tenen un ordre explícit i, per tant, només validaràn si l'ordre en què es defineixen els elements és idèntic a l'ordre en què apareixeran en el document XML.

Per tant, si agafem com a referència el document següent:

```

1 <menjar>
2   <dinar>Arròs</dinar>
3   <sopar>Sopa</sopar>
4 </menjar>
```

Si l'element <menjar> es defineix amb la seqüència (<dinar>, <sopar>), aquest validarà, ja que els elements que conté estan en el mateix ordre que en el document.

```
1 <!ELEMENT menjar (dinar,sopar)>
```

Però no validarà, en canvi, si definíssim la seqüència al revés (<sopar>, <dinar>) perquè el processador esperarà que arribi primer un <sopar> i es trobarà un <dinar>.

```
1 <!ELEMENT menjar (sopar,dinar)>
```

De vegades, en crear documents XML, succeeix que en funció del contingut que hi hagi en el document pot ser que hagin d'aparèixer unes etiquetes o unes altres.

Si dissenyéssim un XML per definir les fitxes de personal d'una empresa podríem tenir una etiqueta <personal> i després definir el càrrec amb una altra etiqueta. El president es podria definir així:

```

1 <personal>
2   <president>Josep Maria Flaviol</president>
3 </personal>
```

I un dels treballadors així:

```

1 <personal>
2   <treballador>Pere Vila</treballador>
3 </personal>
```

Podem fer que una DTD validi tots dos casos fent servir **l'operador d'alternativa** (|).

L'operador alternativa (|) permet que el processador pugui triar entre una de les opcions que se li ofereixen per validar un document XML.

Per tant, podem validar els dos documents XML anteriors definint `<personal>` d'aquesta manera:

```
1 <!ELEMENT personal (treballador|president)>
```

No hi ha cap limitació definida per posar alternatives. Per tant, en podem posar tantes com ens facin falta.

```
1 <!ELEMENT personal (treballador|president|informàtic|gerent)>
```

També es permet mesclar la definició amb `EMPTY` per indicar que el valor pot aparèixer o no:

```
1 <!ELEMENT alumne (delegat|EMPTY)>
```

Combinar alternatives amb `PCDATA` és més complex. Vegeu l'àpartat "Problemes en barrejar etiquetes i `#PCDATA`".

No hi ha res que impedeixi barrejar les seqüències i les alternatives d'elements a l'hora de definir un element amb DTD. Per tant, es poden fer les combinacions que facin falta entre seqüències i alternatives.

Si tenim un XML que defineix l'etiqueta `<cercle>` a partir de les seves coordenades del centre i del radi o el diàmetre podem definir l'element combinant les seqüències amb una alternativa. L'etiqueta `<cercle>` contindrà sempre dins seu les etiquetes `<x>`, `<y>` i després pot contenir també `<radi>` o `<diàmetre>`:

```
1 <!ELEMENT cercle (x,y,(radi|diametre))>
```

Combinar seqüències i alternatives permet saltar-se les restriccions d'ordre de les seqüències. L'exemple següent ens permet definir una persona a partir de nom i cognom en qualsevol ordre:

```
1 <!ELEMENT persona ((cognom,nom)|(nom,cognom))>
```

En els casos en què les etiquetes es repeteixin un nombre indeterminat de vegades serà impossible especificar-les totes en la definició de l'element. **Els modificadors** serveixen per especificar quantes instàncies dels elements fills hi pot haver en un element (taula 2.3).

**TAULA 2.3.** Modificadors acceptats en el contingut dels elements

Modificador	Significat
?	Indica que l'element tant pot ser que hi sigui com no.
+	Es fa servir per indicar que l'element ha de sortir una vegada o més.
*	Indica que pot ser que l'element estigui repetit un nombre indeterminat de vegades, o bé no ser-hi.

Per tant, si volem especificar que una persona pot ser identificada per mitjà del nom

i un o més cognoms podríem definir l'element <persona> d'aquesta manera:

```
1 <!ELEMENT persona (nom,cognom+)>
```

Com que aquesta expressió indica que cognom ha de sortir una vegada, podrà validar tant aquest exemple:

```
1 <persona>
2   <nom>Joan</nom>
3   <cognom>Puig</cognom>
4 </persona>
```

com aquest altre:

```
1 <persona>
2   <nom>Joan</nom>
3   <cognom>Puig</cognom>
4   <cognom>Garcia</cognom>
5 </persona>
```

És evident que aquesta no és la millor manera de definir el que volíem, ja que definit d'aquesta manera també ens acceptarà persones amb 3 cognoms, 4 cognoms, o més.

Per tant, el modificador ideal per forçar que només hi pugui haver un o dos cognoms seria ?, emprant-lo de la manera següent:

```
1 <!ELEMENT persona (nom, cognom, cognom?)>
```

El modificador \* aplicat al mateix exemple ens permetria acceptar que alguna persona no tingüés cognoms o que en tingui un nombre indeterminat:

```
1 <!ELEMENT persona (nom, cognom*)>
```

Els modificadors aplicats darrere d'un parèntesi impliquen aplicar-los a tots els elements de dintre dels parèntesis:

```
1 <!ELEMENT escriptor ((llibre,data)*|articles+)>
```

## Contingut barrejat

El contingut barrejat es va pensar per poder definir continguts que continguin text que es va intercalant amb d'altres elements, com per exemple:

```
1 <carta>Benvolgut <empresa>Ferros Puig</empresa>:
2
3 Sr. <director>Manel Garcia</director>, li envio aquesta carta per comunicar-li
4   que li hem enviat la seva comanda <comanda>145</comanda> a l'adreça que
5   ens va proporcionar.
6
7 Atentament, <empresa>Ferreteria, SA</empresa>
```

El **contingut barrejat** es defineix definint el tipus #PCDATA (sempre en primer lloc) i després s'afegeixen els elements amb l'ajuda de l'operador d'alternativa, |, i s'acaba tot el grup amb el modificador \*.

```
1 <!ELEMENT carta (#PCDATA|empresa|director|comanda)*>
```

S'ha de tenir en compte que aquest contingut només serveix per controlar que els elements hi puguin ser però que no hi ha cap manera de controlar en quin ordre apareixeran els diferents elements.

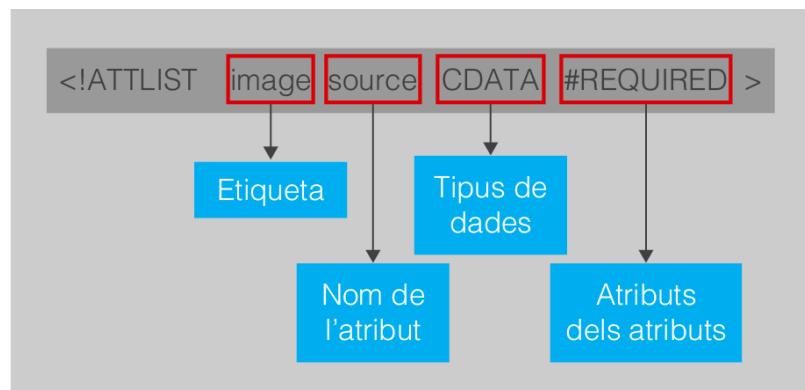
Per tant, podem definir una DTD que validi l'exemple presentat al principi d'aquest apartat de la manera següent:

```
1 <!ELEMENT carta (#PCDATA|empresa|director )*>
2 <!ELEMENT empresa (#PCDATA)>
3 <!ELEMENT director(#PCDATA)>
```

## Atributs en DTD

En les DTD s'han d'especificar quins són els atributs que es faran servir en cada una de les etiquetes explícitament. La declaració d'atributs es fa amb l'etiqueta especial ATTLIST, que es defineix tal com es veu en la figura 2.10.

**FIGURA 2.10.** Definició d'un atribut en DTD



Els dos primers valors de la definició de l'atribut són el **nom de l'etiqueta** i el **nom de l'atribut**, ja que en definir un atribut sempre s'especifica a quina etiqueta pertany. Una de les crítiques que s'han fet a les DTD ha estat que no s'hi poden fer atributs genèrics. Si algú vol definir un atribut que sigui compartit per tots els elements del seu vocabulari ha d'anar especificant l'atribut per a tots i cada un dels elements.

Per definir un atribut anomenat nom que pertanyi a l'element <persona> ho podem fer d'aquesta manera:

```
1 <!ATTLIST persona nom CDATA #IMPLIED>
```

## Especificar múltiples atributs

Si un element necessita diversos atributs haurem d'especificar tots els atributs

en diverses línies ATTLIST. Per exemple, definim els atributs nom i cognoms de l'element <persona> d'aquesta manera:

```
1  <!ATTLIST persona nom CDATA #REQUIRED>
2  <!ATTLIST persona cognom CDATA #REQUIRED>
```

O bé es pot fer la definició dels dos atributs amb una sola referència ATTLIST:

```
1  <!ATTLIST persona nom    CDATA #REQUIRED
2      cognom CDATA #REQUIRED>
```

Les dues declaracions anteriors ens permetrien validar els atributs de l'element <persona> d'aquest exemple:

```
1  <persona nom="Frederic" cognom="Pi" />
```

## Atributs d'ATTLIST

Els elements ATTLIST a part de tipus de dades també poden tenir atributs que permeten definir característiques sobre els atributs. Aquests atributs es poden veure en la taula 2.4.

**TAULA 2.4.** Atributs d'ATTLIST

Atribut	Significat
#IMPLIED	L'atribut és opcional. Els elements el poden tenir o no.
#REQUIRED	L'atribut és obligatori. L'element l'ha de tenir definit o no validarà.
#FIXED	Es fa servir per definir atributs que tenen valors constants i immutables. S'ha d'especificar, ja que és permanent.
#DEFAULT	Permet especificar valors per defecte en els atributs.

Per tant, si es defineix un atribut d'aquesta manera:

```
1  <!ATTLIST equip posicio ID #REQUIRED>
```

s'està obligant que quan es defineixi l'element <equip> en un document XML s'especifiqui obligatòriament l'atribut posicio i que a més el seu valor no es repeteixi en el document, ja que és de tipus ID.

En canvi, definint l'atribut DNI de <persona> amb #IMPLIED es permetrà que en crear el document XML l'atribut DNI hi pugui ser o no.

```
1  <!ATTLIST persona dni NMTOKEN #IMPLIED>
```

En definir un atribut com a #FIXED o com a #DEFAULT se li ha d'especificar el valor. Aquest valor s'especifica a continuació de l'atribut i ha d'anar entre cometes:

```
1  <!ATTLIST document versio CDATA #FIXED "1.0">
2  <!ATTLIST document codificacio NMTOKEN #DEFAULT "UTF-8">
```

Els atributs de tipus #FIXED han de tenir el valor especificat en la definició i aquest no es pot canviar, mentre que els valors definits amb #DEFAULT sí que poden ser canviats.

### Tipus de dades

El tercer paràmetre d'una declaració ATTLIST serveix per definir quins tipus de dades pot tenir l'atribut. Els atributs en DTD no fan servir els tipus de dades dels elements sinó que en defineixen de propis. Els tipus de dades dels atributs es poden veure a la taula 2.5.

**TAULA 2.5.** Tipus de dades dels atributs d'una DTD

Tipus	Significat
CDATA	Pot contenir qualsevol cadena de caràcters acceptable. Es pot fer servir per a preus, URL, adreces electròniques, etc.
Enumeracions	Es fan servir per definir que l'atribut ha de tenir un dels valors específicats.
ID	L'atribut es podrà fer servir com a identificador d'un element. El seu valor serà únic.
IDREF o IDREFS	El valor són referències a un ID que ha d'exsistir. El plural és per definir que és una llista.
ENTITY o ENTITIES	Les entitats permeten definir constants per al document i, per tant, el valor de l'atribut ha de ser una entitat.
NMOKEN o NMOKENS	Especifiquen cadenes de caràcters que només tinguin caràcters permesos per XML. Per tant, no es permeten els espais.
NOTATION	Permet que l'atribut sigui d'una notació declarada anteriorment.

### CDATA

El tipus CDATA és un tipus de dades pràcticament idèntic al tipus #PCDATA de les etiquetes. En un CDATA es pot posar qualsevol dada en format de text tant si té espais com si no en té.

Per tant, la declaració següent:

```
1  <!ATTLIST empresa nom CDATA #REQUIRED>
```

permetria definir etiquetes empresa com aquestes:

```
1  <empresa nom="Microsoft Corporation"/>
2  <empresa nom="6tem"/>
```

És important recordar que els noms també validarien amb un tipus CDATA perquè interpretats en forma de text no deixen de ser simplement caràcters:

```
1  <empresa nom="12"/>
```

## Enumeracions

Els atributs també poden ser especificats definint-hi quins poden ser els valors correctes per a un atribut. Aquests valors s'especifiquen literalment amb l'operador de selecció:

```
1 <!ATTLIST mòdul inici (setembre|febrer) #IMPLIED>
```

Per tant, segons la definició, l'atribut `inici` ha d'existir i només pot tenir els valors “setembre” o “febrer”.

Es poden posar múltiples opcions en una enumeració. Per exemple, podem comprovar que el valor de l'atribut sigui un nombre entre 1 i 12 d'aquesta manera:

```
1 <!ATTLIST mòdul nombre(1|2|3|4|5|6|7|8|9|10|11|12)>
```

## ID

El tipus de dades `ID` serveix per definir atributs que es puguin usar com a identificadors d'un element dins del document. Cal tenir en compte que:

- Els valors assignats no es poden repetir dins del document. Això els fa ideals per fer servir el tipus `ID` per identificar únicament els elements d'un document.
- Els valors han de començar per una lletra o un subratllat.

Els valors de l'atribut `posicio` de la definició següent no es podran repetir dins del mateix document:

```
1 <!ATTLIST equip posicio ID #REQUIRED>
```

Per tant, si es fes servir la declaració anterior per validar aquest document es generaria un error de validació en l'atribut `posicio`, ja que es repeteix el valor “primer” en els dos elements.

```
1 <classificació>
2   <equip posicio="primer">F.C.Barcelona</equip>
3   <equip posicio="primer">Reial Madrid</equip>
4 </classificació>
```

Perquè validi aquest document cal assegurar-se que els atributs `ID` no tinguin el mateix valor:

```
1 <classificació>
2   <equip posicio="primer">F.C.Barcelona</equip>
3   <equip posicio="segon">Reial Madrid</equip>
4 </classificació>
```

## IDREF / IDREFS

Els valors `IDREF` i `IDREFS` es fan servir per definir valors d'atributs que fan

referència a elements que tinguin un valor de tipus ID. Només tenen sentit en vocabularis que tinguin atributs amb valor ID.

IDREF es fa servir per fer referència a un valor ID:

```
1  <!ATTLIST recepta id ID #REQUIRED>
2  <!ATTLIST ingredient ref IDREF #REQUIRED>
```

Amb la declaració que s'acaba d'especificar es pot validar un document com el següent:

```
1  <llibre-cuina>
2      <receptes>
3          <recepta id="recepta1">Patates fregides</recepta>
4          <recepta id="recepta2">Patates bullides</recepta>
5      </receptes>
6      <ingredients>
7          <ingredient ref="recepta1">Oli</ingredient>
8          <ingredient ref="recepta2">Aigua</ingredient>
9      </ingredients>
10 </llibre-cuina>
```

IDREFS permet especificar una llista de valors ID en el mateix atribut. Per exemple, si es defineix l'atribut *ingredient* amb IDREFS:

```
1  <!ATTLIST ingredient ref IDREFS #REQUIRED>
```

Es podria posar una llista d'ID com a valor de l'atribut:

```
1  <ingredient ref="recepta1 recepta2">Patates</ingredient>
```

## NMTOKEN / NMTOKENS

Vegeu l'apartat "Noms vàlids XML" en aquesta unitat.

Els tipus NMTOKEN permeten especificar que els atributs poden tenir qualsevol caràcter acceptat per l'XML.

Així, la declaració següent:

```
1  <!ATTLIST home naixement NMTOKEN #REQUIRED>
```

permetrà validar aquest element:

```
1  <home naixement="1970" />
```

Però no ho farà amb aquest altre, perquè té un espai:

```
1  <home naixement="segle XX" />
```

El valor en plural NMTOKENS permet especificar una llista de valors en comptes d'un de sol:

```
1  <coordenades posicio="x y"/>
```

## NOTATION

Es fa servir per permetre valors que han estat declarats com a *notation* amb l'etiqueta `<!NOTATION>`. Es fa servir per especificar dades no-XML.

Sovint es fa servir per declarar tipus MIME:

```

1  <!NOTATION GIF SYSTEM "image/gif">
2  <!NOTATION JPG SYSTEM "image/jpeg">
3  <!NOTATION PNG SYSTEM "image/png">
4  <!ATTLIST persona
5    photo_type NOTATION (GIF | JPG | PNG) #IMPLIED>
```

## ENTITY / ENTITIES

Indica que el valor és una referència a un valor extern que no s'ha de processar. En general solen contenir valors binaris externs.

Fer servir ENTITY implicarà haver declarat l'entitat amb `<!ENTITY>`:

```

1  <!ATTLIST persona foto ENTITY #IMPLIED
2  <!ENTITY pere SYSTEM "Pere.jpg">
```

Permetrà que es defineixi l'atribut persona amb el valor de l'entitat i que automàticament sigui associat a la imatge:

```
1  <persona nom="pere" />
```

## Altres

Les etiquetes `<!ELEMENT>` i `<!ATTLIST>` no són les úniques que es poden fer servir per declarar DTD. N'hi ha algunes més que en determinats casos es poden fer servir per crear una DTD.

En la taula 2.6 se'n poden veure algunes, i també un breu resum de què és el que fan.

**TAULA 2.6.** Altres etiquetes possibles en una DTD

Etiqueta	Es fa servir per ...
<code>&lt;!ENTITY&gt;</code>	Definir referències a fitxers externs que no s'han de processar.
<code>&lt;!NOTATION&gt;</code>	Incloure dades que no siguin XML en el document.
<code>&lt;!INCLUDE&gt;</code>	Fer que parts de la DTD s'afegeixin al processament.
<code>&lt;!IGNORE&gt;</code>	Fer que parts de la DTD siguin ignorades pel processador.

### 2.3.3 Limitacions

Una de les crítiques que s'ha fet a l'ús de DTD per definir llenguatges XML és que no està basat en XML. La DTD no segueix la manera de definir els documents d'XML i, per tant, per fer-lo servir, **cal aprendre un nou llenguatge**. Si la DTD estigués basada en XML no caldría conèixer de quina manera s'han de definir els elements, marcar les repeticions, els elements buits, etc., ja que es faria amb elements.

Tot i així, el fet que DTD no sigui un llenguatge XML és un problema menor si es tenen en compte les altres limitacions que té:

- No comprova els tipus
- Presenta problemes en barrejar etiquetes i #PCDATA
- Només accepta expressions deterministes

#### La DTD no comprova els tipus

Un dels problemes més importants que ens trobarem a l'hora d'usar DTD per definir el nostre vocabulari és que no té cap manera de comprovar els tipus de dades que contenen els elements. Sovint els noms dels elements ja determinen que el contingut que hi haurà serà d'un tipus determinat (un nombre, una cadena de caràcters, una data, etc.) però la DTD no deixa que se li especifiqui quin tipus de dades s'hi vol posar.

Per tant, si algú emplena amb qualsevol cosa un element que es digui <dia>, el document serà vàlid a pesar que el contingut no sigui una data:

<sup>1</sup> <dia>xocolata</dia>

En no poder comprovar el tipus del contingut, una limitació important afegida és que no hi ha cap manera de poder posar-hi restriccions. Per exemple, no podem definir que volem que una data estigui entre els anys 1900 i 2012.

#### Problemes en barrejar etiquetes i #PCDATA

Una limitació més complexa de veure és que no es poden barrejar etiquetes i #PCDATA en expressions si el resultat no és el que es coneix com a “**contingut barrejat**”.

Vegeu l'apartat “Contingut barrejat” d'aquesta unitat.

Un exemple d'això consistiria a fer una definició d'un exercici com un enunciat de text, però que hi pugui haver diversos apartats que també continguin text.

<sup>1</sup> <exercici>  
<sup>2</sup> Llegiu el text "Validació de documents XML" i respondeu les preguntes següents:  
<sup>3</sup> <apartat numero="1">Què volen dir les sigles XML?</apartat>

```

4 <apartat numero="2">Què és un DTD?</apartat>
5 </exercici>
```

El més senzill seria mesclar un #PCDATA i l'etiqueta <apartat>, però és incorrecte:

```
1 <!ELEMENT exercici (#PCDATA | apartat*)>
```

A més, no es permet que es facin declaracions duplicades d'elements, o sigui que tampoc no ho podem arreglar amb:

```

1 <!ELEMENT exercici(#PCDATA)>
2 <!ELEMENT exercici(apartat*)>
```

L'única manera de combinar-ho seria fer servir la fórmula del contingut barrejat:

```
1 <!ELEMENT exercici(#PCDATA|apartat)*>
```

### Només accepta expressions deterministes

Una altra de les limitacions de les DTD és que obliga que les expressions hagin de ser sempre deterministes.

Si ens mirem un document DTD:

```

1 <!ELEMENT classe(professor|alumnes)>
2 <!ELEMENT professor (nom,cognoms)>
3 <!ELEMENT alumnes (alumne*) >
4 <!ELEMENT alumne (nom,cognom) >
5 <!ELEMENT nom (#PCDATA)>
6 <!ELEMENT cognom (#PCDATA)>
```

Quan s'analitza un fitxer XML es defineix quina és l'arrel de la DTD. Si considerem que l'arrel és l'element <classe>, el procés de validació començarà en la primera línia que ens diu que perquè el document sigui vàlid després de l'arrel hi ha d'haver un element <professor> o bé un element <alumnes>. Si el que arriba és un <professor> el procediment de validació passarà a avaluar la segona i si arriba un <alumne> passarà a la tercera. Si seguim amb l'avaluació veurem que realment el validador sempre que es troba amb una alternativa acabarà anant a una sola expressió. Això és perquè la DTD analitzada és **determinista**.

El mateix ho podem fer amb una expressió més complexa que contingui diferents elements dins de l'alternativa. El validador ha de poder triar, en llegir el primer element, amb quina de les alternatives s'ha de quedar. Si m'arriba un element <nom> em quedo amb l'alternativa de l'esquerra, nom, cognoms, i si m'arriba un <àlies> em quedo amb la de la dreta, àlies, nom, cognoms.

```
1 <!ELEMENT persona(nom,cognom|àlies,nom,cognom)>
```

Si en algun moment algú crea un document que no sigui determinista la validació fallarà. Això passarà si en algun moment el validador es troba que no pot decidir quina és l'alternativa correcta.

<sup>1</sup> `<!ELEMENT terrestre(persona, home | persona, dona)>`

Quan des de l'element `<terrestre>` ens arribi un element `<persona>` el processador no tindrà cap manera de determinar si estem fent referència a l'element `<persona>` de l'expressió de la dreta `persona,home` o bé el de l'esquerra `persona,dona`, i per tant fallarà perquè té dues opcions possibles. És una expressió **no determinista**.

Sovint les expressions no deterministes les podem expressar d'una altra manera, de manera que esdevinguin deterministes. L'exemple anterior es podia haver escrit d'una manera determinista perquè en arribar un element `<persona>` no hi hagi dubtes.

<sup>1</sup> `<!ELEMENT terrestre(persona,(home|dona))>`

Es força que sempre aparegui un element `<persona>` i després hi haurà l'alternativa entre un `<home>` o un `<dona>` que és determinista.

Les expressions no deterministes són més corrents del que sembla, ja que els modificadors les poden provocar i pot ser que no ho semblin. Si es volgués escriure una expressió que determini un llibre a un llibre a partir de l'autor o el títol en qualsevol ordre:

<sup>1</sup> `<!ELEMENT llibre(autor?,titol?|titol?,autor?)>`

Però l'expressió anterior és incorrecta, ja que si el validador es troba un `<autor>` no sap si és l'autor del costat dret de la condició `autor?,titol?` o bé és el del costat esquerre que no té `titol,titol?,autor?`.

L'expressió determinista idèntica a l'anterior seria:

<sup>1</sup> `<!ELEMENT llibre(autor,titol?|titol,autor?|EMPTY)>`

### 2.3.4 Exemple de creació d'una DTD

Les DTD es continuen fent servir perquè són senzilles de crear però cal recordar sempre les limitacions que tenen, i tenir present que no sempre s'adaptaran perfectament a allò que es vol fer.

#### Enunciat

Una empresa ha preparat una botiga d'Internet que genera les comandes dels clients en un format XML que s'envia al programa de gestió de manera automàtica.

Els XML que es generen contenen dades del client i de la comanda que s'ha fet:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE comanda SYSTEM "comanda.dtd">
3 <comanda numero="26" dia="2011-12-01" >
4   <client codi="20">
5     <nom>Frederic</nom>
6     <cognom>Garcia</cognom>
7   </client>
8   <articles>
9     <article>
10       <descripció>Yoda Mimobot USB Flash Drive 8GB</descripció>
11       <quantitat>5</quantitat>
12       <preu>38.99</preu>
13     </article>
14     <article>
15       <descripció>Darth Vader Half Helmet Case for iPhone</descripció>
16       <quantitat>2</quantitat>
17       <preu>29.95</preu>
18     </article>
19   </articles>
20   <total valor="254,85"/>
21 </comanda>
```

Abans de passar-lo al programa han decidit que per tenir més seguretat es farà un pas previ que consistirà a validar que el document. Per això necessiten que es generi la DTD.

## Resolució

S'hauran de fer dues coses:

- Associar les regles al fitxer XML.
- Crear un fitxer amb les regles, que s'anomenarà “comanda.dtd”.

### Associar el fitxer XML

En la segona línia del document s'hi especifica la regla DOCTYPE per associar el fitxer amb la DTD.

```

1 <?xml version="1.0"?>
2 <!DOCTYPE albarà SYSTEM "comanda.dtd">
```

### Crear les regles

No hi ha una manera única de crear una DTD però normalment seguir un sistema pot ajudar a evitar errors. El sistema que es farà servir per resoldre el sistema consisteix a començar per l'arrel i anar definint elsfulls per nivells.

L'arrel del document és l'element <comanda>, que té tres fills:

```

1 <!ELEMENT comanda (client, articles, total)>
```

També té dos atributs, `numero` i `dia`, que només poden tenir valors sense espais i, per tant, seran NMTOKEN. Són dades obligatòries per motius fiscals.

```

1  <!ATTLIST comanda numero NMTOKEN #REQUIRED>
2  <!ATTLIST comanda dia NMTOKEN #REQUIRED>
```

Un cop definit el primer nivell podem passar a definir els altres. D'una banda l'element <client>, que té un atribut per als clients existents i no en tindrà per als nous:

```

1  <!ELEMENT client (nom, cognom)>
2  <!ATTLIST client codi NMTOKEN #IMPLIED >
```

D'altra banda l'element <total>, que és buit però té un atribut:

```

1  <!ELEMENT total EMPTY>
2  <!ATTLIST total valor CDATA #REQUIRED >
```

També l'element <articles>, que contindrà una llista dels articles que ha comprat el client. Com que no tindria sentit fer una comanda sense articles el modificador que es farà servir serà +.

```

1  <!ELEMENT articles (article+)>
```

Per la seva part, desenvolupar <article> tampoc no portarà gaires problemes:

```

1  <!ELEMENT article (descripció, quantitat, preu)>
```

Per acabar només queden els elements que contenen dades:

```

1  <!ELEMENT nom (#PCDATA)>
2  <!ELEMENT cognom (#PCDATA)>
3  <!ELEMENT descripció (#PCDATA)>
4  <!ELEMENT quantitat (#PCDATA)>
5  <!ELEMENT preu (#PCDATA)>
```

El fitxer resultant serà:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!ELEMENT comanda (client, articles, total) >
3  <!ATTLIST comanda numero NMTOKEN #REQUIRED >
4  <!ATTLIST comanda dia NMTOKEN #REQUIRED>
5
6  <!ELEMENT client (nom,cognom) >
7  <!ATTLIST client codi NMTOKEN #IMPLIED >
8
9  <!ELEMENT total EMPTY>
10 <!ATTLIST total valor CDATA #REQUIRED >
11
12 <!ELEMENT articles (article+)>
13 <!ELEMENT article (descripció, quantitat, preu)>
14
15 <!ELEMENT nom (#PCDATA)>
16 <!ELEMENT cognom (#PCDATA)>
17 <!ELEMENT descripció (#PCDATA)>
18 <!ELEMENT quantitat (#PCDATA)>
19 <!ELEMENT preu (#PCDATA)>
```

## 2.4 W3C XML Schema Definition Language

En l'especificació XML es fa referència a les DTD com a mètode per definir vocabularis XML, però les DTD tenen una sèrie de limitacions que van portar el W3C a definir una nova especificació. Aquesta especificació va rebre el nom de **W3C XML Schema Definition Language** (que popularment s'anomena **XML Schema** o **XSD**), i es va crear per substituir la DTD com a mètode de definició de vocabularis per a documents XML.

Es pot trobar l'especificació més recent a <http://www.w3.org/XML/Schema>.

### Relax NG

Durant l'especificació molts membres del grup de treball van considerar que el que sortia era massa complex i van desenvolupar alternatives més simples, entre les quals destaca **Relax NG** (<http://www.relaxng.org>).

Com que **Relax NG** es va fer més tard que DTD i XSD, va poder solucionar alguns dels problemes.

Els esquemes es poden definir en dues versions de Relax NG, una basada en XML i una altra que no ho està, anomenada *Relax NG Compact*, i que està pensada per generar documents d'esquemes més petits i compactes.

L'èxit d'XSD ha estat molt gran i actualment es fa servir per a altres tasques a part de simplement validar XML. També es fa servir en altres tecnologies XML com XQuery, serveis web, etc.

Les característiques més importants que aporta XSD són:

- Està escrit en XML i, per tant, no cal aprendre un llenguatge nou per definir esquemes XML.
- Té el seu propi sistema de dades, de manera que es podrà comprovar el contingut dels elements.
- Suporta espais de noms per permetre barrejar diferents vocabularis.
- Permet ser reutilitzat i segueix els models de programació com herència d'objectes i substitució de tipus.

### 2.4.1 Associar un esquema a un fitxer XML

A diferència del que passa en altres llenguatges de definició –com per exemple en les DTD, en què l'associació s'ha d'especificar en el document XML–, per validar un XML amb un XSD no cal modificar el fitxer XML. De totes maneres, també és possible fer-ho definint-hi l'espai de noms.

Per associar un document XML a un document d'esquema cal definir-hi l'espai de noms amb l'atribut `xmlns`, i per mitjà d'un dels atributs del llenguatge definir-hi

quin és el fitxer d'esquema:

```

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

Es poden definir les referències al fitxer d'esquema de dues maneres, que podem veure en la taula ??.

**TAULA 2.7.** Definir referències a un esquema

Atribut	Significat
<code>noNamespaceSchemaLocation</code>	S'empra quan no es faran servir espais de noms en el document.
<code>schemaLocation</code>	S'empra quan es fan servir explícitament els noms dels espais de noms en les etiquetes.

## 2.4.2 Definir un fitxer d'esquema

L'XSD està basat en XML i, per tant, ha de complir amb les regles d'XML:

- Tot i que no és obligatori normalment sempre es comença el fitxer amb la declaració XML.
- Només hi ha un element arrel, que en aquest cas és `<schema>`.

Com que no s'està generant un document XML lliure sinó que s'està fent servir un vocabulari concret i conegut per poder fer servir els elements XML sempre s'hi haurà d'especificar l'espai de noms d'XSD: “<http://www.w3.org/2001/XMLSchema>”.

```

1 <?xml version="1.0" ?>
2 <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 ...
4 </xsschema>
```

En la declaració d'aquest espai de noms s'està definint que `xs` és l'àlies per fer referència a totes les etiquetes d'aquest espai de noms. Els àlies per a XSD normalment són `xs` o `xsd`, però en realitat no importa quin es faci servir sempre que es faci servir el mateix en tot el document.

L'etiqueta `<schema>` pot tenir diferents atributs, alguns dels quals podem veure en la taula 2.8.

**TAULA 2.8.** Atributs de l'etiqueta “`<schema>`”

Atribut	Significat
<code>attributeFormDefault</code>	Pot ser <code>qualified</code> si fem que sigui obligatori posar l'espai de noms davant dels atributs o <code>unqualified</code> si no ho fem. Per defecte es fa servir <code>unqualified</code> .
<code>elementFormDefault</code>	Serveix per definir si cal l'espai de noms davant del nom. Pot prendre els valors <code>qualified</code> i <code>unqualified</code> .
...	...

### Etiquetes d'XSD

L'XSD defineix moltes etiquetes i no es veuran totes aquí. Podeu trobar totes les etiquetes possibles en l'especificació <http://www.w3.org/TR/xmlschema11-1/>.

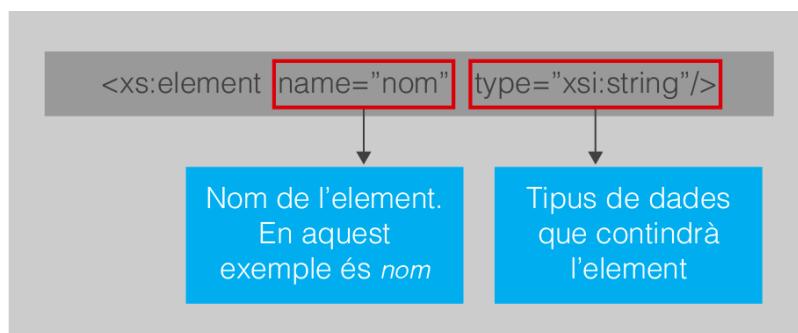
**TAULA 2.8** (continuació)

Atribut	Significat
version	Permet definir quina versió del document d'esquemes estem definint (no és la versió d'XML Schemas).

A partir de l'etiqueta arrel ja es poden començar a definir les etiquetes del vocabulari que es vol crear.

### Definició d'elements

Els elements es defineixen tal com es veu a la figura 2.11, fent servir l'etiqueta <element>, i amb atributs s'hi especifica com a mínim el nom i opcionalment el tipus de dades que contindrà l'element.

**FIGURA 2.11.** Definició d'un element

L'XSD divideix els elements en dos grans grups basant-se en les dades que contenen:

- **Elements amb contingut de tipus simple:** són elements sense atributs que només contenen dades.
- **Elements amb contingut de tipus complex:** són elements que poden tenir atributs, no tenir contingut o contenir elements.

A partir de la definició es pot veure que gairebé sempre hi haurà algun tipus complex, ja que l'arrel normalment contindrà altres elements.

### Elements amb contingut de tipus simple

Es consideren elements amb contingut de tipus simple aquells que no contenen altres elements ni tenen atributs.

La versió 1.1 d'XSD defineix una cinquantena de tipus de dades diferents, que es poden trobar a la seva definició (<http://www.w3.org/TR/xmlschema11-2/>). Entre els més usats en destaquen els de la taula 2.9:.

**TAULA 2.9.** Tipus de dades més usats en XSD

Tipus	Dades que s'hi poden emmagatzemar
<i>string</i>	Cadenes de caràcters
<i>decimal</i>	Valors numèrics
<i>boolean</i>	Només pot contenir 'true' o 'false' o (1 o 0)
<i>date</i>	Dates en forma (AAAA-MM-DD)
<i>anyURI</i>	Referències a llocs (URL, camins de disc...)
<i>base64binary</i>	Dades binàries codificades en <i>base64</i>
<i>integer</i>	Nombres enters

A partir dels tipus bàsics, l'estàndard en crea d'altres amb l'objectiu de tenir tipus de dades que es puguin adaptar millor als objectius de qui dissenya l'esquema. En aquest sentit apareixen els tipus anomenats *positiveInteger*, *nonNegativeInteger*, *gYearMonth*, *unsignedInt*...

Els tipus de dades permeten restringir els valors que contindran les etiquetes XML. Per exemple, si es parteix de la definició següent:

```

1 <xsd:element name="posicio" type="xsd:integer"/>
2 </xsd:schema>
```

Només s'aconseguirà validar un element si el seu contingut és un nombre enter. Per exemple, l'exemple següent no validarà:

```

1 <posicio>Primer</posicio>
```

En la taula 2.10 es poden veure exemples de definicions d'elements i valors que hi validen.

**TAULA 2.10.** Exemples d'elements i què s'hi pot validar

Etiqueta	Exemple
<xsd:element name="dia" type="xsd:date" />	<dia>2011-09-15</dia>
<xsd:element name="alçada" type="xsd:integer"/>	<alçada>220</alçada>
<xsd:element name="nom" type="xsd:string"/>	<nom>Pere Puig</nom>
<xsd:element name="mida" type="xsd:float"/>	<mida>1.7E2</mida>
<xsd:element name="lloc" type="xsd:anyURI"/>	<lloc> <a href="http://www.ioc.cat">http://www.ioc.cat</a> </lloc>

Quan es defineix una etiqueta en XSD s'està definint que l'etiqueta haurà de sortir en el document XML una vegada. És bastant habitual que hi hagi etiquetes que es repeteixin un nombre determinat de vegades. En XSD això s'ha simplificat per mitjà d'uns atributs de l'etiqueta *<element>* que determinen la cardinalitat dels elements:

- **minOccurs:** permet definir quantes vegades ha de sortir un element com a mínim. Un valor de '0' indica que l'element pot no sortir.
- **maxOccurs:** serveix per definir quantes vegades com a màxim pot sortir

un element. `unbounded` implica que no hi ha límit en les vegades que pot sortir.

Fent servir els atributs es pot definir que l'element `<nom>` ha de sortir una vegada i l'element `<cognom>` un màxim de dues vegades.

```
1 <xss:element name="nom" />
2 <xss:element name="cognom" maxOccurs="2"/>
```

També es poden donar valors als elements amb els atributs `fixed`, `default` i `nullable`.

L'atribut `fixed` permet definir un valor obligatori per a un element:

```
1 <xss:element name="centre" type="xs:string" fixed="IOC"/>
```

De manera que només es podrà definir el contingut amb el valor especificat (o sense res):

```
1 <centre />
2 <centre>IOC</centre>
```

Però mai un valor diferent de l'especificat:

```
1 <centre>Institut Cendrassos</centre>
```

A diferència de `fixed`, `default` assigna un valor per defecte però deixa que sigui canviat en el contingut de l'etiqueta.

```
1 <xsi:element name="centre" type="xs:string" default="IOC" />
```

La definició permetria validar amb els tres casos següents:

```
1 <centre />
2 <centre>IOC</centre>
3 <centre>Institut Cendrassos</centre>
```

L'atribut `nullable` es fa servir per dir si es permeten continguts nuls. Per tant, només pot prendre els valors `yes` o `no`.

## Tipus simples personals

Com que a vegades pot interessar definir valors per als elements que no han de coincidir necessàriament amb els estàndards, l'XSD permet definir tipus de dades personals. Per exemple, si es vol un valor numèric però que no accepti tots els valors sinó un subconjunt dels enters.

Per definir tipus simples personals no es posa el tipus a l'element i s'hi defineix un fill `<simpleType>`.

```
1 <xss:element name="persona">
2   <xss:simpleType>
3     ...
4     </xss:simpleType>
5   </xss:element>
```

A pesar que es poden definir llistes de valors no es recomana gaire fer-ne servir. La majoria dels experts creuen que és millor definir els valors de la llista fent servir repeticions d'etiquetes.

Dins de **simpleType** s'especifica quina és la modificació que s'hi vol fer. El més corrent és que les modificacions sigui fetes amb **list**, **union**, **restriction** o **extension**.

Fer servir **list** permetrà definir que un element pot contenir llistes de valors. Per tant, per especificar que un element **<partits>** pot contenir una llista de dates es definiria:

```

1 <xss:element name="partits">
2   <xss:simpleType>
3     <xss:list itemType="xss:date"/>
4   </xss:simpleType>
5 </xss:element>

```

L'etiqueta validaria amb una cosa com:

```

1 <partits> 2011-01-07 2011-01-15 2011-01-21</partits>

```

Els elements **simpleType** també es poden definir amb un nom fora dels elements i posteriorment usar-los com a tipus de dades personal.

```

1 <xss:simpleType name="dies">
2   <xss:list itemType="xss:date"/>
3 </xss:simpleType>
4
5 <xss:element name="partits" type="dies"/>

```

Fent servir els tipus personalitzats amb nom es poden crear modificacions de tipus **union**. Els modificadors **union** serveixen per fer que es pugui mesclar tipus diferents en el contingut d'un element.

La definició de l'element **<preu>** farà que l'element pugui ser de tipus **valor** o de tipus **simbol**.

```

1 <xss:element name="preus">
2   <xss:simpleType>
3     <xss:union memberTypes="valor simbol"/>
4   </xss:simpleType>
5 </xss:element>

```

Amb això li podríem assignar valors com aquests:

```

1 <preus>25 €</preus>

```

Però sense cap mena de dubte el modificador més interessant és el que permet definir restriccions als tipus base. Amb el modificador **restriction** es poden crear tipus de dades en què només s'acceptin alguns valors, que les dades compleixin una determinada condició, etc.

L'element **<naixement>** només podrà tenir valors enters entre 1850 i 2011 si es defineix d'aquesta manera:

```

1 <xss:simpleType name="any_naixement">
2   <xss:restriction base="xss:integer">
3     <xss:maxInclusive value="2011"/>

```

```

4     <xss:minInclusive value="1850"/>
5   </xss:restriction>
6 </xss:simpleType>
7
8 <xss:element name="naixement" type="any_naixement"/>
```

Es poden definir restriccions de molts tipus per mitjà d'atributs (taula 2.11). Normalment els valors de les restriccions s'especifiquen en l'atribut `value`:

**TAULA 2.11.** Atributs que permeten definir restriccions en XSD

Elements	Resultat
<code>maxInclusive</code> / <code>maxExclusive</code>	Es fa servir per definir el valor numèric màxim que pot agafar un element.
<code>minInclusive</code> / <code>minExclusive</code>	Permet definir el valor mínim del valor d'un element.
<code>length</code>	Amb <code>length</code> restringim la llargada que pot tenir un element de text. Podem fer servir <code>&lt;xss:minLength&gt;</code> i <code>&lt;xss:maxLength&gt;</code> per ser més precisos.
<code>enumeration</code>	Només permet que l'element tingui algun dels valors especificats en les diferents línies <code>&lt;enumeration&gt;</code> .
<code>totalDigits</code>	Permet definir el nombre de díigits d'un valor numèric.
<code>fractionDigits</code>	Serveix per especificar el nombre de decimals que pot tenir un valor numèric.
<code>pattern</code>	Permet definir una expressió regular a la qual el valor de l'element s'ha d'adaptar per poder ser vàlid.

Per exemple, el valor de l'element `<resposta>` només podrà tenir un dels tres valors “A”, “B” o “C” si es defineix d'aquesta manera:

```

1 <xss:element name="resposta">
2   <xss:simpleType>
3     <xss:enumeration value="A"/>
4     <xss:enumeration value="B"/>
5     <xss:enumeration value="C"/>
6   </xss:simpleType>
7 </xss:element>
```

Una de les restriccions més interessants són les definides per l'atribut `pattern`, que permet definir restriccions a partir d'expressions regulars. Com a norma general tenim que si s'especifica un caràcter en el patró aquest caràcter ha de sortir en el contingut; les altres possibilitats les podem veure a la taula 2.12

**TAULA 2.12.** Definició d'expressions regulars

Símbol	Equivalència
.	Qualsevol caràcter
\d	Qualsevol dígit
\D	Qualsevol caràcter no dígit
\s	Caràcters no imprimibles: espais, tabuladors, salts de línia...
\S	Qualsevol caràcter imprimible
x*	El de davant de * ha de sortir 0 o més vegades
x+	El de davant de + ha de sortir 1 o més vegades
...	...

**TAULA 2.12** (continuació)

Símbol	Equivalència
x?	El de davant de ? pot sortir o no
[abc]	Hi ha d'haver algun caràcter dels de dins
[0-9]	Hi ha d'haver un valor entre el dos especificats, amb aquests inclosos
x{5}	Hi ha d'haver 5 vegades el que hi hagi al davant dels claudàtors
x{5,}	Hi ha d'haver 5 o més vegades el de davant
x{5,8}	Hi ha d'haver entre 5 i 8 vegades el de davant

Fent servir aquest sistema es poden definir tipus de dades molt personalitzats. Per exemple, podem definir que una dada ha de tenir la forma d'un DNI (8 xifres, un guió i una lletra majúscula) amb aquesta expressió:

```

1 <xss:simpleType name="dni">
2   <xss:restriction base="xss:string">
3     <xss:pattern value="[0-9]{8}-[A-Z]" />
4   </xss:restriction>
5 </xss:simpleType>
```

A part de les restriccions també hi ha l'element **extension**, que serveix per afegir característiques extra als tipus. No té gaire sentit que surti en elements de tipus simple però es pot fer servir, per exemple, per afegir atributs als tipus simples (cosa que els converteix en tipus complexos).

### Elements amb contingut de tipus complex

Els elements amb contingut de tipus complex són aquells que tenen atributs, contenen altres elements o no tenen contingut.

Els elements amb contingut complex han rebut moltes crítiques perquè es consideren massa complicats, però s'han de fer servir perquè en tots els fitxers d'esquema normalment hi haurà un tipus complex: l'arrel del document.

Es considera que hi ha quatre grans grups de continguts de tipus complex:

1. Els que en el seu contingut només tenen dades. Per tant, són com els de tipus simples però amb atributs.
2. Els elements que en el contingut només tenen elements.
3. Els elements buits.
4. Els elements amb contingut barrejat.

Els elements amb tipus complex es defineixen especificant que el tipus de dades de l'element és `<xss:complexType>`.

```

1 <xs:element name="classe">
2   <xs:complexType>
3     ...
4   </xs:complexType>
5 </xs:element>
```

De la mateixa manera que amb els tipus simples, es poden definir tipus complexos amb nom per reutilitzar-los com a tipus personalitzats.

```

1 <xs:complexType name="curs">
2   ...
3 </xs:complexType>
4
5 <xs:element classe type="curs"/>
```

## Atributs

Una característica bàsica d'XSD és que només els elements de tipus complex poden tenir atributs. En essència no hi ha gaires diferències entre definir un element o un atribut, ja que es fa de la mateixa manera però fent servir l'etiqueta **attribute**.

Els tipus de dades són els mateixos i, per tant, poden tenir tipus bàsics com en l'exemple següent:

```
1 <xs:attribute name="número" type="xs:integer" />
```

S'hi poden posar restriccions de la mateixa manera que en els elements. En aquest exemple l'atribut **any** no pot tenir valors superiors a 2011 si es defineix d'aquesta manera:

```

1 <xs:attribute name="any">
2   <xs:simpleType>
3     <xs:restriction base="xs:integer">
4       <xs:maxInclusive="2011"/>
5     </xs:restriction>
6 </xs:attribute>
```

Tret que s'especifiqui el contrari, els atributs sempre són opcionals.

L'etiqueta **<attribute>** té una sèrie d'atributs que permeten definir característiques extra sobre els atributs ([taula 2.13](#)).

**TAULA 2.13.** Atributs més importants de **xs:attribute**

Atribut	Ús
<b>use</b>	Permet especificar si l'atribut és obligatori ( <b>required</b> ), opcional ( <b>optional</b> ) o bé no es pot fer servir ( <b>prohibited</b> ).
<b>default</b>	Defineix un valor per defecte.
<b>fixed</b>	Es fa servir per definir valors obligatoris per als atributs.
...	...

**TAULA 2.13** (continuació)

Atribut	Ús
<code>form</code>	Permet definir si l'atribut ha d'anar amb l'àlies de l'espai de noms ( <code>qualified</code> ) o no ( <code>unqualified</code> ).

Per exemple, l'atribut `any` s'haurà d'especificar obligatòriament si es defineix de la manera següent:

```
1 <xs:attribute name="any" type="xs:integer" use="required" />
```

### 'simpleContent'

Si l'element només conté text, el contingut de `complexType` serà un `simpleContent`. El `simpleContent` permet definir restriccions o extensions a elements que només tenen dades com a contingut.

La diferència més important és que en aquest cas es poden definir atributs en l'element. Els atributs s'afegeixen definint una extensió al tipus fet servir en l'element.

En aquest exemple l'element `<Mida>` té contingut de tipus enter i defineix dos atributs, `llargada` i `amplada`, que també són enters.

```
1 <xs:complexType name="Mida">
2   <xs:simpleContent>
3     <xs:extension base="xs:integer">
4       <xs:attribute name="llargada" type="xs:integer"/>
5       <xs:attribute name="amplada" type="xs:integer"/>
6     </xs:extension>
7   </xs:simpleContent>
8 </xs:complexType>
```

### Contingut format per elements

Els elements que contenen altres elements també poden ser definits en XSD dins d'un `<complexType>` i poden ser alguns dels elements de la taula 2.14.

**TAULA 2.14.** Elements per definir contingut complex

Etiqueta	Serveix per
<code>sequence</code>	Especificar el contingut com una llista ordenada d'elements.
<code>choice</code>	Permet especificar elements alternatius.
<code>all</code>	Definir el contingut com una llista desordenada d'elements.
<code>complexContent</code>	Estendre o restringir el contingut complex.

L'element `<sequence>` és una de les maneres amb què el llenguatge XSD permet que s'especifiquin els elements que han de formar part del contingut d'un element. Fins i tot en el cas en què només hi hagi una sola etiqueta es pot definir com a una seqüència.

La condició més important que tenen és que **els elements del document XML**

**per validar han d'aparèixer en el mateix ordre** en el qual es defineixen en la seqüència.

```

1 <xs:element name="persona">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="nom" type="xs:string"/>
5       <xs:element name="cognom" type="xs:string" maxOccurs="2"/>
6       <xs:element name="tipus" type="xs:string" />
7     </xs:sequence>
8   </xs:complexType>
9 </xs:element>
```

En l'exemple anterior es defineix que abans de l'aparició de `<tipus>` poden aparèixer un o dos cognoms.

```

1 <persona>
2   <nom>Marcel</nom>
3   <cognom>Puig</cognom>
4   <cognom>Lozano</cognom>
5   <tipus>Professor</tipus>
6 </persona>
```

No validarà cap contingut si algun element no està exactament en el mateix ordre.

```

1 <persona>
2   <tipus>Professor</tipus>
3   <cognom>Puig</cognom>
4   <nom>Marcel</nom>
5 </persona>
```

Les seqüències poden contenir dins seu altres seqüències d'elements.

```

1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
2   <xs:element name="persona">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element name="nomcomplet">
6           <xs:complexType>
7             <xs:sequence>
8               <xs:element name="nom" type="xs:string"/>
9               <xs:element name="cognom" type="xs:string" maxOccurs="2"/>
10            </xs:sequence>
11          </xs:complexType>
12        </xs:element>
13        <xs:element name="professió" type="xs:string"/>
14      </xs:sequence>
15    </xs:complexType>
16  </xs:element>
17 </xs:schema>
```

L'element `<choice>` serveix per fer que s'hagi de triar una de les alternatives de les que es presenten dins seu.

En aquest exemple l'element persona podrà contenir o bé l'etiqueta `<nomCognoms>` o bé `<dni>`, però no totes dues.

```

1 <xs:complexType name="persona">
2   <xs:choice>
3     <xs:element name="nomCognoms" type="xs:string"/>
```

```

4     <xss:element name="dni" type="xs:string"/>
5   </xss:choice>
```

Entre les alternatives hi pot haver seqüències o altres elements `<choice>`. La definició següent és un exemple més elaborat que l'anterior i permet que es pugui triar entre els elements `<nom>` i `<cognom>` o `<dni>`.

```

1 <xss:choice>
2   <xss:sequence>
3     <xss:element name="nom" type="xs:string"/>
4     <xss:element name="cognom" type="xs:string" maxOccurs="2"/>
5   </xss:sequence>
6   <xss:element name="dni" type="xs:string"/>
7 </xss:choice>
```

La diferència més important entre l'element `<all>` i `<sequence>` és l'ordre. L'element `<all>` permet especificar una seqüència d'elements però permet que s'especifiquin en qualsevol ordre.

Per tant, si definim l'element `<persona>` de la manera següent:

```

1 <xss:element name="persona">
2   <xss:complexType>
3     <xss:all>
4       <xss:element name="nom"/>
5       <xss:element name="cognom"/>
6     </xss:all>
7   </xss:complexType>
8 </xss:element>
```

Ens servirà per validar tant aquest document:

```

1 <persona>
2   <nom>Pere</nom>
3   <cognom>Garcia</cognom>
```

com aquest:

```

1 <persona>
2   <cognom>Garcia</cognom>
3   <nom>Pere</nom>
```

Però sempre s'han de tenir en compte les limitacions d'aquest element que no eren presents en les seqüències ordenades:

- Dins seu només hi pot haver elements. No hi pot haver ni seqüències, ni alternatives.
- No es pot fer servir la cardinalitat en els elements que contingui, ja que provocaria un problema de *no-determinisme*.

Per tant, l'exemple següent no és correcte, ja que es demana que `<cognom>` pugui sortir dues vegades.

```

1 <xss:all>
2   <xss:element name="nom" type="xs:string"/>
3   <xss:element name="cognom" maxOccurs="2" type="xs:string"/>
4 </xss:all>
```

Una possible manera de permetre que es pugui especificar el nom i els dos cognoms en qualsevol ordre seria fer el següent:

```

1 <xss:complexType>
2   <xss:choice>
3     <xss:sequence>
4       <xss:element name="nom" type="xss:string"/>
5       <xss:element name="cognom" type="xss:string" maxOccurs="2"/>
6     </xss:sequence>
7     <xss:sequence>
8       <xss:element name="cognom" type="xss:string" maxOccurs="2"/>
9       <xss:element name="nom" type="xss:string"/>
10    </xss:sequence>
11  </xss:choice>
12 </xss:complexType>
```

### 'complexContent'

L'etiqueta `complexContent` permet definir *extensions* o *restriccions* a un tipus complex que contingui contingut barrejat o només elements.

Això fa que amb una extensió es pugui ampliar un contingut complex ja existent o restringir-ne els continguts.

Per exemple, si ja hi ha definit un tipus de dades `nomCompleto` en què hi ha els elements `<nom>` i `<cognom>` se'n pot reutilitzar la definició per definir un nou tipus de dades, `agenda`, en què s'afegeixi l'adreça de correu electrònic.

```

1 <xss:complexType name="nomCompleto">
2   <xss:sequence>
3     <xss:element name="nom" type="xss:string"/>
4     <xss:element name="cognom" type="xss:string" maxOccurs="2"/>
5   </xss:sequence>
6 </xss:complexType>

7
8 <xss:complexType name="agenda">
9   <xss:complexContent>
10    <xss:extension base="nomCompleto">
11      <xss:sequence>
12        <xss:element name="email" type="xss:string" />
13      </xss:sequence>
14    </xss:extension>
15  </xss:complexContent>
16 </xss:complexType>
```

D'aquesta manera es podrà definir un element de tipus `agenda`:

```
1 <xss:element name="persona" type="agenda"/>
```

que haurà de tenir els tres elements `<nom>`, `<cognom>`, i `<email>`:

```

1 <persona>
2   <nom>Pere</nom>
3   <cognom>Garcia</cognom>
4   <email>pgarcia@ioc.cat</email>
5 </persona>
```

### Elements sense contingut

Per a XSD els elements sense contingut són sempre de tipus complex. En la

definició simplement no s'especifica cap contingut i tindrem un element buit.

```

1 <xss:element name="delegat">
2   <xss:complexType />
3 </xss:element>
```

La definició permet definir l'element d'aquesta manera:

```

1 <delegat />
```

Si l'element necessita atributs simplement s'especifiquen dins del `complexType`.

```

1 <xss:element name="delegat">
2   <xss:complexType>
3     <xss:attribute name="any" use="required" type="xs:gYear"/>
4   </xss:complexType>
5 </xss:element>
```

I ja es podrà definir l'atribut en l'element buit:

```

1 <delegat any="2012"/>
```

### Contingut barrejat

Els elements amb contingut barrejat són els elements que tenen de contingut tant elements com text. Es va pensar per poder incloure elements enmig d'un text narratiu.

En XSD el contingut barrejat es defineix posant l'atribut `mixed="true"` en la definició del l'element `<complexType>`.

```

1 <xss:element name="carta">
2   <xss:complexType mixed="true">
3     <xss:sequence>
4       <xss:element name="nom" type="xs:string"/>
5       <xss:element name="dia" type="xs:gDay"/>
6     </xss:sequence>
7   </xss:complexType>
8 </xss:element>
```

Això ens permetria validar un contingut com aquest:

```

1 <carta>Estimat senyor <nom>Pere<nom>:
2
3 Li envio aquesta carta per recordar-li que hem quedat per
4 trobar-nos el dia <dia>12</dia>
5 </carta>
```

### 2.4.3 Exemple de creació d'un XSD

Es poden crear definicions de vocabularis XSD a partir de la idea del que volem que continguin les dades o bé a partir d'un fitxer XML de mostra.

## Enunciat

En un centre en què només s'imparteixen els cicles formatius d'SMX i ASIX, quan han d'entrar les notes als alumnes ho fan creant un arxiu XML com el següent:

```

1  <?xml version="1.0" ?>
2  <classe modul="3" nommodul="Programació bàsica">
3      <curs especialitat="ASIX">1</curs>
4      <professor>
5          <nom>Marcel</nom>
6          <cognom>Puig</cognom>
7      </professor>
8      <alumnes>
9          <alumne>
10             <nom>Filomeno</nom>
11             <cognom>Garcia</cognom>
12             <nota>5</nota>
13         </alumne>
14         <alumne delegat="si">
15             <nom>Frederic</nom>
16             <cognom>Pi</cognom>
17             <nota>3</nota>
18         </alumne>
19         <alumne>
20             <nom>Manel</nom>
21             <cognom>Puigdevall</cognom>
22             <nota>8</nota>
23         </alumne>
24     </alumnes>
25 </classe>
```

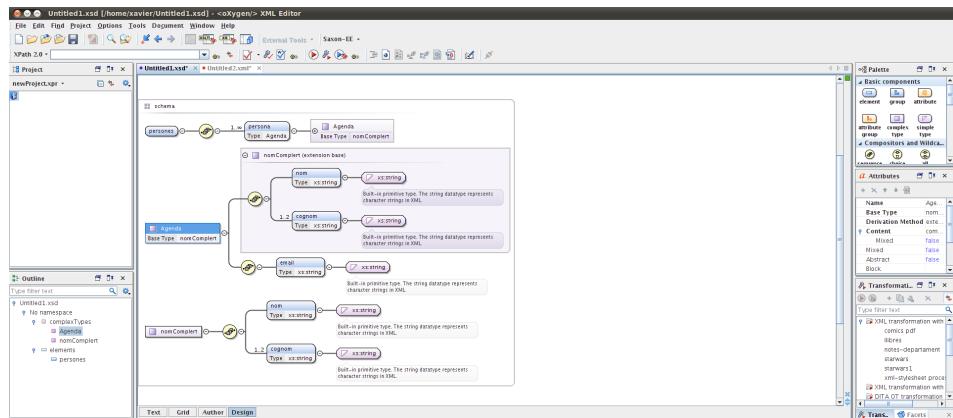
A la secretaria necessiten que es generi la definició del vocabulari en XSD per comprovar que els fitxers que reben són correctes.

## Resolució

A l'hora de dissenyar un esquema des d'un XML sempre s'ha de partir d'un fitxer XML que tingui totes les opcions que poden sortir en cada element, o el resultat no serà vàlid per a tots els fitxers.

Una de les coses que s'han de tenir en compte a l'hora de fer un exercici com aquest és que no hi ha una manera única de fer-lo. Es pot fer amb tipus personalitzats, sense tipus personalitzats, a vegades es pot resoldre combinant uns elements però també amb uns altres, etc. Per tant, la solució que s'ofерirà aquí és només una de les possibles solucions.

Una altra cosa que s'ha de tenir en compte és que els editors XML soLEN oferir una manera gràfica de crear XSD que permet crear esquemes de manera més senzilla, com es pot veure en la figura 2.12.

**FIGURA 2.12.** Creació d'esquemes en mode disseny de l'oXygen XML Editor

## Anàlisi

La primera fase normalment consisteix a analitzar el fitxer per determinar si hi ha parts que poden ser susceptibles de formar un tipus de dades. En l'exercici es pot veure que tant per al professor com per als alumnes les dades que contenen són semblants (els alumnesafegeixen la nota), i per tant es pot crear un tipus de dades que farà més simple el disseny final.

Per tant, en l'arrel podem crear el tipus complex persona, que contindrà el nom i el cognom.

```

1 <xs:complexType name="persona">
2   <xs:sequence>
3     <xs:element name="nom" type="xs:string"/>
4     <xs:element name="cognom" type="xs:string"/>
5   </xs:sequence>
6 </xs:complexType>
```

Com que els alumnes són iguals però amb un element extra es potaprofitar el tipus persona estenent-lo.

```

1 <xs:complexType name="estudiant">
2   <xs:complexContent>
3     <xs:extension base="persona">
4       <xs:sequence>
5         <xs:element name="nota">
6           ...
7         </xs:element>
8       </xs:sequence>
9     </xs:extension>
10   </xs:complexContent>
11 </xs:complexType>
```

Les notes no poden tenir tots els valors (només d'1 a 10), o sigui, que es pot afegir una restricció al tipus enter.

```

1 <xs:element name="nota">
2   <xs:simpleType>
3     <xs:restriction base="xs:integer">
4       <xs:maxInclusive value="10"/>
5       <xs:minInclusive value="1"/>
6     </xs:restriction>
7   </xs:simpleType>
8 </xs:element>
```

## Desenvolupament

L'arrel sempre serà de tipus complex perquè conté atributs i tres elements, de manera que es pot començar la declaració com una seqüència d'elements.

```

1 <xss:element name="classe">
2   <xss:complexType>
3     <xss:sequence>
4       <xss:element name="curs">
5         ...
6         </xss:element>
7       <xss:element name="professor" type="persona"/>
8       <xss:element name="alumnes">
9         ...
10        </xss:element>
11      </xss:sequence>
12    </xss:complexType>
13 </xss:element>
```

S'han deixat els elements `<curs>` i `<alumnes>` per desenvolupar-los, mentre que l'element `<professor>` ja es pot tancar perquè s'ha definit el tipus persona.

L'element `<curs>` no té contingut i, per tant, serà de tipus complex. A més, s'hi han de definir dos atributs.

```

1 <xss:element name="curs">
2   <xss:complexType>
3     <xss:attribute name="numero" use="required">
4       ...
5     </xss:attribute>
6     <xss:attribute name="especialitat" use="required">
7       ...
8     </xss:attribute>
9   </xss:complexType>
10 </xss:element>
```

Els atributs s'han de desenvolupar perquè no poden tenir tots els valors:

- `numero` només pot ser “1” o “2”.
- `especialitat` serà “SMX” o “ASIX”.

La manera més adequada per fer-ho serà restringir els valors amb una enumeració.

```

1 <xss:attribute name="numero" use="required">
2   <xss:simpleType>
3     <xss:restriction base="xs:integer">
4       <xss:enumeration value="1"/>
5       <xss:enumeration value="2"/>
6     </xss:restriction>
7   </xss:simpleType>
8 </xss:attribute>
9 <xss:attribute name="especialitat" use="required">
10  <xss:simpleType>
11    <xss:restriction base="xs:string">
12      <xss:enumeration value="ASIX"/>
13      <xss:enumeration value="SMX"/>
14    </xss:restriction>
15  </xss:simpleType>
16 </xss:attribute>
```

Ja només queda per desenvolupar `<alumnes>`, que té una repetició d'elements `<alumne>`, del qual ja n'hi ha un tipus.

```

1 <xs:element name="alumne">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="alumne" type="estudiant"
5         maxOccurs="unbounded" minOccurs="1"/>
6     </xs:sequence>
7   </xs:complexType>
8 </xs:element>

```

El resultat final serà:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3
4   <xs:complexType name="persona">
5     <xs:sequence>
6       <xs:element name="nom" type="xs:string"/>
7       <xs:element name="cognom" type="xs:string"/>
8     </xs:sequence>
9   </xs:complexType>
10
11  <xs:complexType name="estudiant">
12    <xs:complexContent>
13      <xs:extension base="persona">
14        <xs:sequence>
15          <xs:element name="nota">
16            <xs:simpleType>
17              <xs:restriction base="xs:integer">
18                <xs:maxInclusive value="10"/>
19                <xs:minInclusive value="1"/>
20              </xs:restriction>
21            </xs:simpleType>
22          </xs:element>
23        </xs:sequence>
24      </xs:extension>
25    </xs:complexContent>
26  </xs:complexType>
27
28  <xs:element name="classe">
29    <xs:complexType>
30      <xs:sequence>
31        <xs:element name="curs">
32          <xs:complexType>
33            <xs:attribute name="numero" use="required">
34              <xs:simpleType>
35                <xs:restriction base="xs:integer">
36                  <xs:enumeration value="1"/>
37                  <xs:enumeration value="2"/>
38                </xs:restriction>
39              </xs:simpleType>
40            </xs:attribute>
41            <xs:attribute name="especialitat" use="required">
42              <xs:simpleType>
43                <xs:restriction base="xs:string">
44                  <xs:enumeration value="ASIX"/>
45                  <xs:enumeration value="SMX"/>
46                </xs:restriction>
47              </xs:simpleType>
48            </xs:attribute>
49          </xs:complexType>
50        </xs:element>
51        <xs:element name="professor" type="persona"/>
52        <xs:element name="alumnes" type="estudiant"/>
53      </xs:sequence>
54    </xs:complexType>
55  </xs:element>
56 </xs:schema>

```

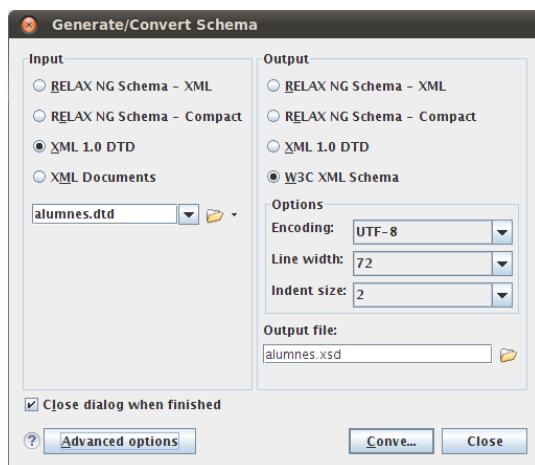
## 2.5 Conversió entre esquemes

Una de les eines que s'han de tenir en compte a l'hora de fer esquemes és que ja hi ha eines que permeten convertir els llenguatges de definició d'esquemes d'un format a un altre.

A pesar d'això sempre s'hauran de revisar els resultats per comprovar que no s'ha perdut cap significat important en fer la conversió, ja que els sistemes automàtics no són perfectes.

La majoria dels editors XML porten alguna eina per fer la conversió (figura 2.13) però també hi ha eines específiques per fer conversions, com el programa de codi obert Trang.

**FIGURA 2.13.** L'oxygen permet convertir els esquemes d'un format a un altre



### 2.5.1 Trang

El Trang és un programa de codi obert que permet convertir les definicions de vocabularis fetes en un sistema a un altre. Pot convertir els llenguatges següents:

- RELAX NG (XML syntax)
- RELAX NG compact syntax
- XML 1.0 DTD
- W3C XML Schema

Per fer la conversió simplement hem d'especificar el fitxer que es vol convertir i en què es vol convertir. L'única condició és que l'esquema d'origen no sigui un XML Schema.

Es pot convertir fent servir fitxers que tinguin les extensions per defecte: DTD (.dtd), XML Schemas (.xsd), RELAX NG (.rng), RELAX NG compact (.rnc).

Amb això es crearà `alumnes.xsd` a partir d'`'alumnes.dtd'`.

```
1 $ trang alumnes.dtd alumnes.xsd
```

O bé especificant explícitament quin és el tipus de fitxer d'entrada (`-I`) i quin el de sortida (`-O`).

```
1 $ trang -I dtd -O xsd alumnes.dtd alumnes.xsd
```

Una de les característiques interessants que té és que pot crear el vocabulari basant-se en fitxers XML de mostra. A partir del fitxer XML, `alumnes.xml`, es pot fer que el Trang generi automàticament l'XML Schema que el validaria executant:

```
1 $ trang alumnes.xml alumnes.xsd
```

Si es tenen diversos documents XML també pot generar l'esquema associat amb aquests:

```
1 $ trang *.xml alumnes.xsd
```