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:

```
<!DOCTYPE ... >
<?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.

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

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

Ni tampoc al final:

```
<?xml version="1.0" ?>
<document>
</document>
<!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:

```
<?xml version="1.0" ?>
<!DOCTYPE ... >
<document>
</document>
```

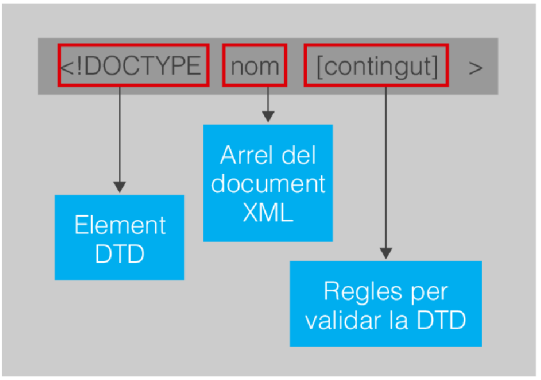
Hi ha dues maneres d'incorporar DTD en un document XML:

- 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`:

```
<?xml version="1.0"?>
<!DOCTYPE classe [
    <!ELEMENT classe (professor, alumnes)>
    <!ELEMENT professor (#PCDATA)>
    <!ELEMENT alumnes (nom*)>
    <!ELEMENT nom (#PCDATA)>
  ]>
<classe>
   <professor>Marcel Puig</professor>
   <alumnes>
```

```
11        <nom>Frededic 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

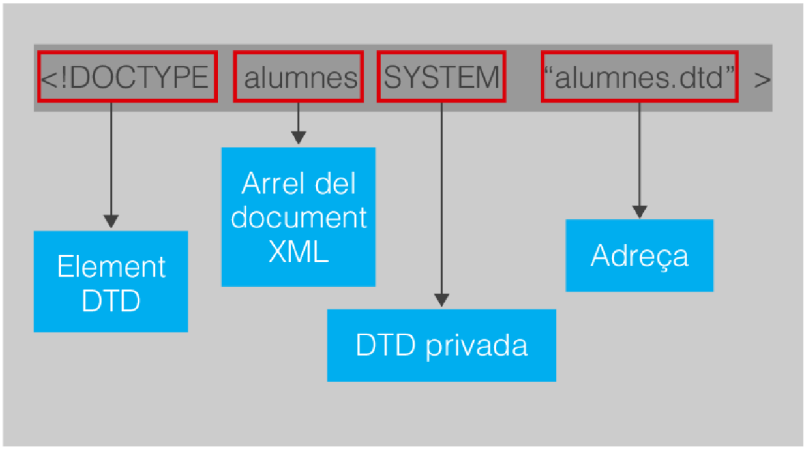
En comptes d'especificar la DTD en el mateix document es considera una pràctica molt millor definir-la en un fitxer a part.

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 comparteixi 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:

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

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.

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

```
<!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:

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

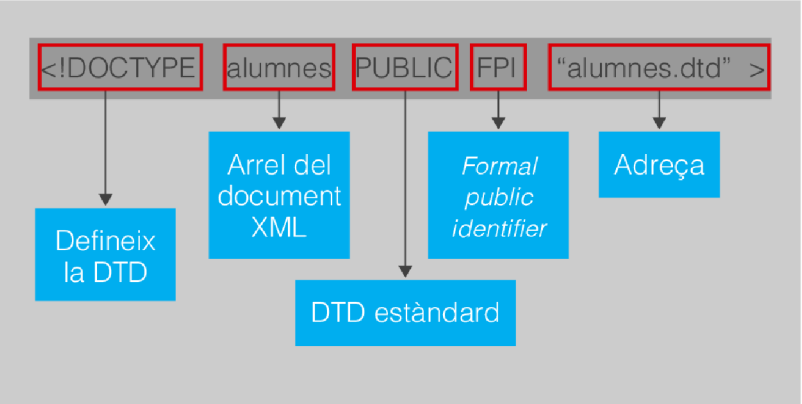
Un cop definit només cal crear l'arxiu `alumnes.dtd` en el lloc adequat amb les regles que defineixen el vocabulari:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT alumne (professor, alumnes) >
<!ELEMENT professor (#PCDATA) >
<!ELEMENT alumnes (nom*) >
<!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:

```
"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:

```
<!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:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"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.

```
<!DOCTYPE nom SYSTEM "fitxer.dtd" [ regles ] >
```

```
<!DOCTYPE classe PUBLIC "-//IOC//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:

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

Com d'aquesta:

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

2.3.2 Definició d'esquemes amb DTD

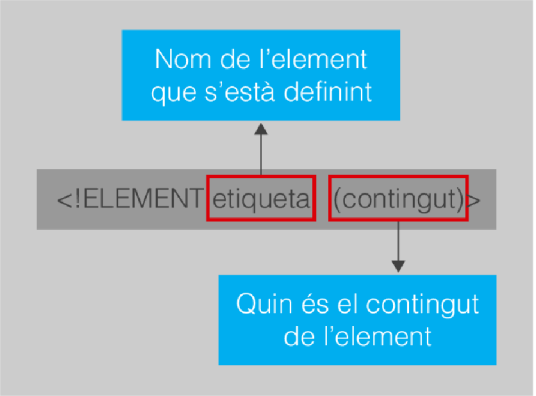
La part més important d'un sistema de definició de vocabularis és definir com es fa per determinar l'ordre en què els elements poden aparèixer i quin contingut poden tenir, quins atributs poden tenir les etiquetes, etc.

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.

```
<!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:

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

validarà tant elements que continguin dades:

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

com elements que continguin altres elements:

```
<persona>
    <nom>Frederic</nom>
    <cognom>Pi</cognom>
</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.

```
</persones>
    <persona>
        <nom>Pere Martí</nom>
        <professor/>
```

```
    </persona>
    <persona>
        <nom>Marcel Peris</nom>
    </persona>
</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:

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

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

```
<professor/>
```

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

```
<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:

```
<persona>
    <nom>Marcel</nom>
    <cognom>Puigdevall</nom>
</persona>
```

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

```
<!ELEMENT nom (#PCDATA)>
<!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>`.

```
<persona>
    <nom>Pere</nom>
    <cognom>Martinez</cognom>
</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.

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

Mai no s'ha d'oblidar que les seqüències tenen un ordre explícit i, per tant, només validaran 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:

```
<menjar>
   <dinar>Arròs</dinar>
   <sopar>Sopa</sopar>
</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.

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

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

```
<!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:

```
<personal>
   <president>Josep Maria Flaviol</president>
</personal>
```

I un dels treballadors així:

```
<personal>
    <treballador>Pere Vila</treballador>
</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:

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

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

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

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

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

Combinar alternatives amb `PCDATA` és més complex. Vegeu l'apartat "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>`:

```
<!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:

```
<!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:

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

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

```
<persona>
    <nom>Joan</nom>
    <cognom>Puig</cognom>
</persona>
```

com aquest altre:

```
<persona>
    <nom>Joan</nom>
    <cognom>Puig</cognom>
    <cognom>Garcia</cognom>
</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:

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

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

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

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

```
<!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:

```
<carta>Benvolgut <empresa>Ferros Puig</empresa>:

Sr. <director>Manel Garcia</director>, li envio aquesta carta per comunicar-li
     que li hem enviat la seva comanda <comanda>145</comanda> a l'adreça que
     ens va proporcionar.

  Atentament, <empresa>Ferreteria, SA</empresa>

</carta>
```

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 *.

```
<!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.

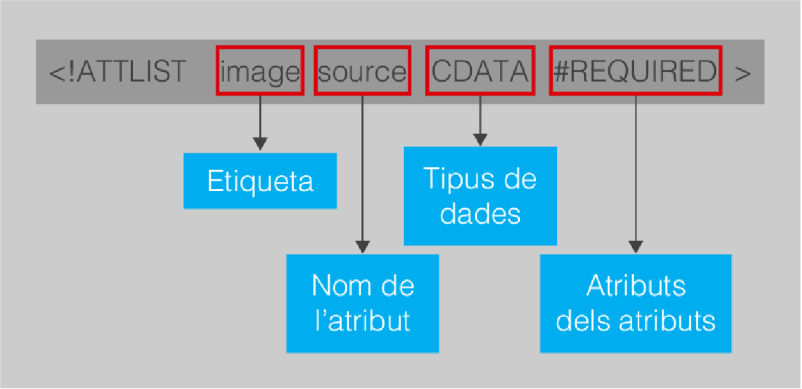
Per tant, podem definir una DTD que validi l'exemple presentat al principi d'aquest apartat de la manera següent:

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

```
<!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:

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

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

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

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

```
<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:

```
<!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.

```
<!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:

```
<!ATTLIST document versio CDATA #FIXED "1.0">
<!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 especificats.
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'existir. 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.
NMTOKEN o NMTOKENS	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:

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

permetria definir etiquetes `empresa` com aquestes:

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

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

```
<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:

```
<!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:

```
<!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:

```
<!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 `posició`, ja que es repeteix el valor "primer" en els dos elements.

```
<classificació>
  <equip posició="primer">F.C.Barcelona</equip>
  <equip posició="primer">Reial Madrid</equip>
</classificació>
```

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

```
<classificació>
  <equip posició="primer">F.C.Barcelona</equip>
  <equip posició="segon">Reial Madrid</equip>
</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`:

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

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

```
<llibre_cuina>
    <receptes>
        <recepta id="recepta1">Patates fregides</recepta>
        <recepta id="recepta2">Patates bullides</recepta>
    </receptes>
    <ingredients>
        <ingredient ref="recepta1">Oli</ingredient>
        <ingredient ref="recepta2">Aigua</ingredient>
    </ingredients>
</llibre_cuina>
```

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

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

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

```
<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:

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

permetrà validar aquest element:

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

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

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

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

```
<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:

```
<!NOTATION GIF SYSTEM "image/gif">
<!NOTATION JPG SYSTEM "image/jpeg">
<!NOTATION PNG SYSTEM "image/png">
<!ATTLIST persona
  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`:

```
<!ATTLIST persona foto ENTITY #IMPLIED
<!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:

```
<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 ...
<!ENTITY>	Definir referències a fitxers externs que no s'han de processar.
<!NOTATION>	Incloure dades que no siguin XML en el document.
<!INCLUDE>	Fer que parts de la DTD s'afegeixin al processament.
<!IGNORE>	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 caldria 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:

```
1 <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.

```
1 <exercici>
2    Llegiu el text "Validació de documents XML" i responeu les preguntes següents:
3    <apartat numero="1">Què volen dir les sigles XML?</apartat>
```

```
   <apartat numero="2">Què és un DTD?</apartat>
</exercici>
```

El més senzill seria mesclar un `#PCDATA` i l'etiqueta `<apartat>`, però és incorrecte:

```
<!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:

```
<!ELEMENT exercici(#PCDATA)>
<!ELEMENT exercici(apartat*)>
```

L'única manera de combinar-ho seria fer servir la fórmula del `contingut barrejat`:

```
<!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:

```
<!ELEMENT classe(professor|alumnes)>
<!ELEMENT professor (nom,cognoms)>
<!ELEMENT alumnes (alumne*) >
<!ELEMENT alumne (nom,cognom) >
<!ELEMENT nom (#PCDATA)>
<!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`.

```
<!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.

```
<!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.

```
<!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:

```
<!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 títol, `titol?,autor?`.

L'expressió determinista idèntica a l'anterior seria:

```
<!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:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE comanda SYSTEM "comanda.dtd">
<comanda numero="26" dia="2011-12-01" >
    <client codi="20">
        <nom>Frederic</nom>
        <cognom>Garcia</cognom>
    </client>
    <articles>
        <article>
            <descripció>Yoda Mimobot USB Flash Drive 8GB</descripció>
            <quantitat>5</quantitat>
            <preu>38.99</preu>
        </article>
        <article>
            <descripció>Darth Vader Half Helmet Case for iPhone</descripció>
            <quantitat>2</quantitat>
            <preu>29.95</preu>
        </article>
    </articles>
    <total valor="254,85"/>
</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.

```
<?xml version="1.0"?>
<!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 els fulls per nivells.

L'arrel del document és l'element `<comanda>`, que té tres fills:

```
<!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.

```
<!ATTLIST comanda numero NMTOKEN #REQUIRED>
<!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:

```
<!ELEMENT client (nom, cognom)>
<!ATTLIST client codi NMTOKEN #IMPLIED >
```

D'altra banda l'element `<total>`, que és buit però té un atribut:

```
<!ELEMENT total EMPTY>
<!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 `+`.

```
<!ELEMENT articles (article+)>
```

Per la seva part, desenvolupar `<article>` tampoc no portarà gaires problemes:

```
<!ELEMENT article (descripció, quantitat, preu)>
```

Per acabar només queden els elements que contenen dades:

```
<!ELEMENT nom (#PCDATA)>
<!ELEMENT cognom (#PCDATA)>
<!ELEMENT descripció (#PCDATA)>
<!ELEMENT quantitat (#PCDATA)>
<!ELEMENT preu (#PCDATA)>
```

El fitxer resultant serà:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT comanda (client, articles, total) >
<!ATTLIST comanda numero NMTOKEN #REQUIRED >
<!ATTLIST comanda dia NMTOKEN #REQUIRED>

<!ELEMENT client (nom,cognom) >
<!ATTLIST client codi NMTOKEN #IMPLIED >

<!ELEMENT total EMPTY>
<!ATTLIST total valor CDATA #REQUIRED >

<!ELEMENT articles (article+)>
<!ELEMENT article (descripció, quantitat, preu)>

<!ELEMENT nom (#PCDATA)>
<!ELEMENT cognom (#PCDATA)>
<!ELEMENT descripció (#PCDATA)>
<!ELEMENT quantitat (#PCDATA)>
<!ELEMENT preu (#PCDATA)>
```