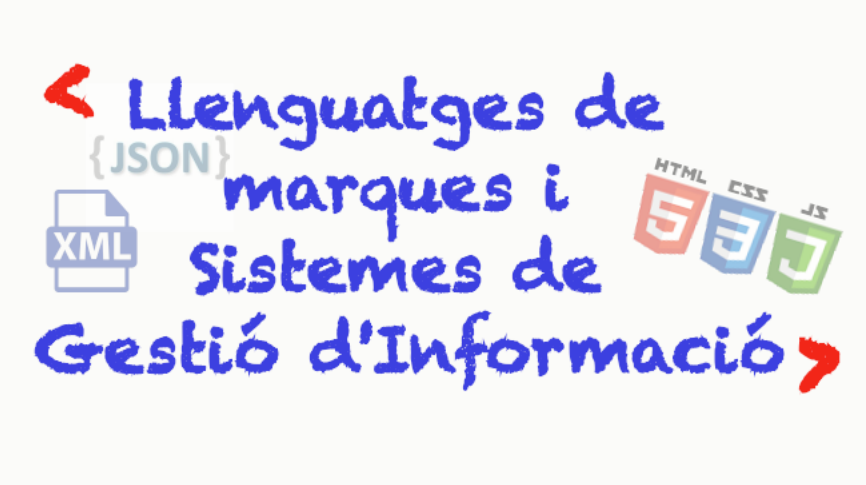
UNITAT 5. DEFINICIÓ D'ESQUEMES I VOCABULARIS EN LLENGUATGES DE MARQUES



IES Sant Vicent Ferrer
Algemesí

Contingut

1. Introducció

L'XML permet crear els llenguatges de marques que vulgam, siga quin siga el camp d'actuació. Aquest és el punt fort de l'XML sobre altres llenguatges de marques: s'adapta al que es vulga representar sense importar la complexitat que puga 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 tan 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>, <cercle>. 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:

```
<dibuix>
            <rectangle>12,10,14,10</rectangle>
            <linia>13,13,25,25</linia>
</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 siguen correctes. Com que aquesta tasca és molt complexa 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**.

1.1. Validació de documents

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

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·laboren 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 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

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 siguen 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.

2. 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 fora 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 Schema. Tot i això encara és molt usat, sobretot perquè és molt més senzill.

2.1. 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 aparega 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 siga just a continuació de la declaració XML, i per tant tampoc no es pot incloure l'etiqueta DOCTYPE un cop 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

2.1.1. Declaració DTD interna

En les declaracions DTD internes les regles de la DTD estan incorporades en el document XML.

Exemple:

```
<?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>
                <nom>Frededic Pi</nom>
        </alumnes>
</classe>
```

2.1.2. 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 defineix el vocabulari com a privat, no hi ha res que llimite que el fitxer es puga compartir o es publique per mitjà d'Internet.

La definición privada queda de la següent forma:

```
<!DOCTYPE elemento-raíz SYSTEM "URI">
```

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

DTD públiques

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

La definició pública queda de la següent forma:

```
<!DOCTYPE elemento-raíz PUBLIC "identificador-público" "URI">
```

2.1.3. 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

<!ELEMENT nom (contingut)>

Continguts genèrics

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 estiguen 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 continguen dades:

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

com elements que continguen altres elements:

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

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

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 continguen 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 puguen 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 DTDs

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.

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.

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.

2.2. Limitacions DTD

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:

```
<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".

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 continguen text.

```
<exercici>
   Llegiu el text "Validació de documents XML" i responeu les preguntes següents:
   <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 esdevinguen 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 determine 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. 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 genere 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 seua part, desenvolupar <article> tampoc no portarà 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)>
```

3. 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 www.w3.org/XML/Schema.

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.

3.1. 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:

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

Es poden definir les referències al fitxer d'esquema de dues maneres:

Atribut	Significat
noNamespaceSchemaLocation	S'empra quan no es faran servir espais de noms en el document.
schemaLocation	S'empra quan es fan servir explícitament els noms dels espais de noms en les etiquetes.

3.2. 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".

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    ...
</xs:schema>
```

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.

Etiquetes d'XSD

L'XSD defineix moltes etiquetes i no es veuran totes aquí. Podeu trobar totes les etiquetes possibles en l'especificació www.w3.org/TR/xmlschema11-1 .

L'etiqueta <schema> pot tenir diferents atributs.

Alguns atributs de l'etiqueta <schema>:

Atribut	Significat
attributeFormDefault	Pot ser qualified si fem que sigui obligatori posar l'espai de noms davant dels atributs o unqualified si no ho fem. Per defecte es fa servir unqualified.
elementFormDefault	Serveix per definir si cal l'espai de noms davant del nom. Pot prendre els valors qualified i unqualified.
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 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.

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 quasi 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 (www.w3.org/TR/xmlschema11-2). Entre els més usats en destaquen:

Tipus	**Dades que s'hi poden emmagatzemar**
string	Cadenes de caràcters
decimal	Valors numèrics
boolean	Només pot contenir 'true' o 'false' o (1 o 0)
date	Dates en forma (AAAA-MM-DD)
anyURI	Referències a llocs (URL, camins de disc…)
base64binary	Dades binàries codificades en base64
integer	Nombres enters

A partir dels tipus bàsics, l'estàndard en crea d'altres amb l'objectiu de tenir tipus de dades que es puguen 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:

```
<xs:element name="posicio" type="xs:integer"/>
</xs:schema>
```

Només s'aconseguirà validar un element si el seu contingut és un nombre enter. Per exemple, l'exemple següent no validarà:

```
<posicio>Primer</posicio>
```

A continuación, es poden veure exemples de definicions d'elements i valors que hi validen.

Etiqueta	**Exemple**
<xs:element name="dia" type="xs:date" />	<dia>2011-09-15</dia>
<xs:element name="alçada" type="xs:integer"/>	<alçada>220</alçada>
<xs:element name="nom" type="xs:string"/>	<nom>Pere Puig</nom>
<xs:element name="mida" type="xs:float"/>	<mida>1.7E2</mida>
<xs:element name="lloc" type="xs:anyURI"/>	<lloc>http://www.ioc.cat</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.

```
<xs:element name="nom" />
<xs: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:

```
<xs:element name="centre" type="xs:string" fixed="IOC"/>
```

De manera que només es podrà definir el contingut amb el valor especificat (o sense res):

```
<centre />
<centre>IOC</centre>
```

Però mai un valor diferent de l'especificat:

```
<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.

```
<xsi:element name="centre" type="xs:string" default="IOC" />
```

La definició permetria validar amb els tres casos següents:

```
<centre />
<centre>IOC</centre>
<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>.

```
<xs:element name="persona">
    <xs:simpleType>
       ...
    </xs:simpleType>
</xs:element>
```

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.

A pesar que es poden definir llistes de valors no es recomana fer-ne servir. La majoria dels experts creuen que és millor definir els valors de la llista fent servir repeticions d'etiquetes.

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:

```
<xs:element name="partits">
  <xs:simpleType>
      <xs:list itemType="xs:date"/>
  </xs:simpleType>
</xs:element>
```

L'etiqueta validaria amb una cosa com:

```
<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.

```
<xs:simpleType name="dies">
    <xs:list itemType="xs:date"/>
</xs:simpleType>
 <xs: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 puguen 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.

```
<xs:element name="preus">
    <xs:sipleType>
        <xs:union memberTypes="valor simbol"/>
    </xs:simpleType>
</xs:element>
```

Amb això li podríem assignar valors com aquests:

```
<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 compleixen una determinada condició, etc.

L'element <naixement> només podrà tenir valors enters entre 1850 i 2011 si es defineix d'aquesta manera:

```
<xs:simpleType name="any_naixement">
    <xs:restricion base="xs:integer">
        <xs:maxInclusive value="2011"/>
        <xs:minInclusive value="1850"/>
    </xs:restriction>
</xs:simpleType>
     <xs:element name="naixement" type="any_naixement"/>
```

Atributs que permeten definir restriccions en XSD

Elements	**Resultat**
maxInclusive / maxExclusive	Es fa servir per definir el valor numèric màxim que pot agafar un element.
minInclusive / minExclusive	Permet definir el valor mínim del valor d'un element.
length	Amb lenght restringim la llargada que pot tenir un element de text. Podem fer servir <xs:minLength> i <xs:maxLenght> per ser més precisos.
enumeration	Només permet que l'element tingui algun dels valors especificats en les diferents línies <enumeration>.
totalDigits	Permet definir el nombre de dígits d'un valor numèric.
fractionDigits	Serveix per especificar el nombre de decimals que pot tenir un valor numèric.
pattern	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:

```
<xs:element name="resposta">
    <xs:simpleType>
        <xs:enumeration value="A"/>
        <xs:enumeration value="B"/>
        <xs:enumeration value="C"/>
    </xs:simpleType>
</xs: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.

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
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:

```
<xs:simpleType name="dni">
    <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{8}-[A-Z]"/>
    </xs:restriction>
</xs:simpleType>
```

A part de les restriccions també hi ha l'element extension, que serveix per afegir característiques extra als tipus. No té sentit que apareguen 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:

- Els que en el seu contingut només tenen dades. Per tant, són com els de tipus simples però amb atributs.
- Els elements que en el contingut només tenen elements.
- Els elements buits.
- Els elements amb contingut barrejat.

Els elements amb tipus complex es defineixen especificant que el tipus de dades de l'element és <xs:complexType>.

```
<xs:element name="classe">
    <xs:complexType>
          ....
    </xs:complexType>
</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.

```
<xs:complexType name="curs">
     ...
</xs:complexType>


<xs:element classe type="curs"/>
```

Atributs

Una característica bàsica d'XSD és que <u>només els elements de tipus complex poden tenir atributs</u>. En essència no hi ha 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:

```
<xs:attribute name="número" type="xs:integer" />
```

Es 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:

```
<xs:attribute name="any">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:maxInclusive value="2011"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
```

Si no s'especifica el contrari, els atributs sempre són opcionals.

L'etiqueta <attribute> té una sèrie d'atributs que permeten definir característiques extra sobre els atributs

Atributs més importants de xs:attribute

Atribut	**Ús**
use	Permet especificar si l'atribut és obligatori (required), opcional (optional) o bé no es pot fer servir (prohibited).
default	Defineix un valor per defecte.
fixed	Es fa servir per definir valors obligatoris per als atributs.
form	Permet definir si l'atribut ha d'anar amb l'àlies de l'espai de noms (qualified) o no (unqualified).

Per exemple, l'atribut any s'haurà d'especificar obligatòriament si es defineix de la manera següent:

```
<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.

```
<xs:complexType name="Mida">
    <xs:simpleContent>
        <xs:extension base="xs:integer">
           <xs:atribute name="llargada" type="xs:integer"/>
           <xs:atribute name="amplada" type="xs:integer"/>
        </xs:extension>
    </xs:simpleContent>
</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 següents:

Etiqueta	**Serveix per**
sequence	Especificar el contingut com una llista ordenada d'elements.
choice	Permet especificar elements alternatius.
all	Definir el contingut com una llista desordenada d'elements.
complexContent	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 haja 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.

```
<xs:element name="persona">
    <xs:complexContent>
        <xs:sequence>
            <xs:element name="nom" type="xs:string"/>
            <xs:element name="cognom" type="xs:string" maxOccurs="2"/>
            <xs:element name="tipus" type="xs:string" />
        </xs:sequence>
    </xs:complexContent>
</xs:element>
```

En l'exemple anterior es defineix que abans de l'aparició de <tipus> poden aparèixer un o dos cognoms.

```
<persona>
    <nom>Marcel</nom>
    <cognom>Puig</cognom>
    <cognom>Lozano</cognom>
    <tipus>Professor</tipus>
</persona>
```

No validarà cap contingut si algun element no està exactament en el mateix ordre.

```
<persona>
    <tipus>Professor</tipus>
    <cognom>Puig</cognom>
    <nom>Marcel</nom>
</persona>
```

Les seqüències poden contenir dins seu altres seqüències d'elements.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="persona">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="nomcomplet">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="nom" type="xs:string"/>
                            <xs:element name="cognom" type="xs:string" maxOccurs="2"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="professió" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</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.

```
<xs:complexType nom="persona">
  <xs:choice>
      <xs:element name="nomCognoms" type="xs:string"/>
      <xs:element name="dni" type="xs:string"/>
  </xs: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>.

```
<xs:choice>
    <xs:sequence>
        <xs:element name="nom" type="xs:string"/>
        <xs:element name="cognom" type="xs:string" maxOccurs="2"/>
    </xs:sequence>
    <xs:element name="dni" type="xs:string"/>
</xs: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:

```
<xs:element name="persona">
    <xs:complexType>
        <xs:all>
            <xs:element name="nom"/>
            <xs:element name="cognom"/>
        </xs:all>
    </xs:complexType>
</xs:element>
```

Ens servirà per validar tant aquest document:

```
<persona>
    <nom>Pere</nom>
    <cognom>Garcia</nom>
```

com aquest:

```
<persona>
    <cognom>Garcia</nom>
    <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> puga eixir dues vegades.

```
<xs:all>
    <xs:element name="nom" type="xs:string"/>
    <xs:element name="cognom" maxOccurs="2" type="xs:string"/>
</xs:all>
```

Una possible manera de permetre que es puguen especificar el nom i els dos cognoms en qualsevol ordre seria fer el següent:

```
<xs:complexType>
    <xs:choice>
        <xs:sequence>
            <xs:element name="nom" type="xs:string"/>
            <xs:element name="cognom" type="xs:string" maxOccurs="2"/>
        </xs:sequence>
        <xs:sequence>
            <xs:element name="cognom" type="xs:string" maxOccurs="2"/>
            <xs:element name="nom" type="xs:string"/>
        </xs:sequence>
    </xs:choice>
</xs:complexType>
```

'complexContent'

L'etiqueta complexContent permet definir extensions o restriccions a un tipus complex que continguen contingut barrejat o només elements.

Això fa que amb una extensió es puga ampliar un contingut complex ja existent o restringir-ne els continguts.

Per exemple, si ja hi ha definit un tipus de dades nomComplet 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.

```
<xs:complexType name="nomComplet">
  <xs:sequence>
      <xs:element name="nom" type="xs:string"/>
      <xs:element name="cognom" type="xs:string" maxOccurs="2"/>
  </xs:sequence>
</xs:complexType>
     <xs:complexType name="agenda">
    <xs:complexContent>
        <xs:extension base="nomComplet">
            <xs:sequence>
                 <xs:element name="email" type="xs:string" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

D'aquesta manera es podrà definir un element de tipus agenda:

```
<xs:element name="persona" type="agenda"/>
```

que haurà de tenir els tres elements <nom>, <cognom>, i <email>:

```
<persona>
    <nom>Pere</nom>
    <cognom>Garcia</cognom>
    <email>pgarcia@ioc.cat</email>
</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.

```
<xs:element name="delegat">
    <xs:complexType />
</xs:element>
```

La definició permet definir l'element d'aquesta manera:

```
<delegat />
```

Si l'element necessita atributs simplement s'especifiquen dins del complexType.

```
<xs:element name="delegat">
    <xs:complexType>
        <xs:attribute name="any" use="required" type="xs:gYear"/>
    </xs:complexType>
</xs:element>
```

I ja es podrà definir l'atribut en l'element buit:

```
<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>.

```
<xs:element name="carta">
    <xs:complexType mixed="true">
        <xs:sequence>
            <xs:element name="nom" type="xs:string"/>
            <xs:element name="dia" type="xs:gDay"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Això ens permetria validar un contingut com aquest:

```
<carta>Estimat senyor <nom>Pere<nom>:
Li envio aquesta carta per recordar-li que hem quedat per
trobar-nos el dia <dia>12</dia>
</carta>
```

3.3. Exemple de creació d'un XSD

Es poden crear definicions de vocabularis XSD a partir de la idea del que volem que continguen 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:

```
<?xml version="1.0" ?>
<classe modul="3" nommodul="Programació bàsica">
    <curs numero="1" especialitat="ASIX">
        <professor>
            <nom>Marcel</nom>
            <cognom>Puig</cognom>
        </professor>
        <alumnes>
            <alumne>
                <nom>Filomeno</nom>
                <cognom>Garcia</cognom>
                <nota>5</nota>
            </alumne>
            <alumne delegat="si">
                <nom>Frederic</nom>
                <cognom>Pi</cognom>
                <nota>3</nota>
            </alumne>
            <alumne>
                <nom>Manel</nom>
                <cognom>Puigdevall</cognom>
                <nota>8</nota>
            </alumne>
        </alumnes>
    </curs>
</classe>
```

A la secretaria necessiten que es genere 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'oferirà aquí és només una de les possibles solucions.

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 alumnes afegeixen 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.

```
<xs:complexType name="persona">
    <xs:sequence>
        <xs:element name="nom" type="xs:string"/>
        <xs:element name="cognom" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

Com que els alumnes són iguals però amb un element extra es pot aprofitar el tipus persona estenent-lo.

```
<xs:complexType name="estudiant">
    <xs:complexContent>
        <xs:extension base="persona">
            <xs:sequence>
                <xs:element name="nota">
                    ...
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</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.

```
<xs:element name="nota">
    <xs:simpleType>
         <xs:restriction base="xs:integer">
                <xs:maxInclusive value="10"/>
                <xs:minInclusive value="1"/>
         </xs:restriction>
    </xs:simpleType>
</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.

```
<xs:element name="classe">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="curs">
                ...
            </xs:element>
            <xs:element name="professor" type="persona"/>
            <xs:element name="alumnes">
                 ...
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs: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.

```
<xs:element name="curs">
    <xs:complexType>
        <xs:attribute name="numero" use="required">
            ...
        </xs:attribute>
        <xs:attribute name="especialitat" use="required">
            ...
        </xs:attribute>
    </xs:complexType>
</xs: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.

```
<xs:attribute name="numero" use="required">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:enumeration value="1"/>
            <xs:enumeration value="2"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
<xs:attribute name="especialitat" use="required">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="ASIX"/>
            <xs:enumeration value="SMX"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
```

Ja només queda per desenvolupar <alumnes>, que té una repetició d'elements <alumne>, del qual ja n'hi ha un tipus.

```
<xs:element name="alumne">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="alumne" type="estudiant"
                        maxOccurs="unbounded" minOccurs="1"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

El resultat final serà:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:complexType name="persona">
    <xs:sequence>
        <xs:element name="nom" type="xs:string"/>
        <xs:element name="cognom" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="estudiant">
    <xs:complexContent>
        <xs:extension base="persona">
            <xs:sequence>
                <xs:element name="nota">
                    <xs:simpleType>
                        <xs:restriction base="xs:integer">
                            <xs:maxInclusive value="10"/>
                            <xs:minInclusive value="1"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="delegat" use="optional">
                <xs:simpleType>
```

```
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="si"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:element name="classe">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="curs">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="professor" type="persona"/>
                            <xs:element name="alumnes">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element              name="alumne"
type="estudiant" maxOccurs="unbounded" />
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                        <xs:attribute name="numero" use="required">
                            <xs:simpleType>
                                <xs:restriction base="xs:integer">
                                    <xs:enumeration value="1"/>
                                    <xs:enumeration value="2"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="especialitat" use="required">
                            <xs:simpleType>
```

```
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="ASIX"/>
                                    <xs:enumeration value="SMX"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="modul" use="required" type="xs:integer" />
            <xs:attribute name="nommodul" use="required" type="xs:string" />
        </xs:complexType>
    </xs:element>
    </xs:schema>
```

4. Enllaços d'interés

DTDs a w3school: https://www.w3schools.com/xml/xml_dtd_intro.asp

5. Bibliografia

Sala, Xavier. (2023) Llenguatges de marques i sistemes de gestió d'informació.

Institut Obert de Catalunya