

Llenguatges de marques

Llenguatges per a l'estructuració
i el tractament de la informació

Francesc Santanach i Delisau

PID_00216118

Índex

Introducció	5
Objectius	6
1. Documents electrònics	7
1.1. Marcatge de format	8
1.2. Marcatge generalitzat	8
2. L'especificació XML	11
2.1. Elements	12
2.2. Atributs	12
2.3. Instruccions de procés	13
2.4. Sintaxi	13
2.5. L'arbre XML	15
2.6. Treballar amb XML	16
2.6.1. Espai de noms	16
2.6.2. <i>Document type definition (DTD)</i>	18
2.6.3. <i>Extensible stylesheet language (XSL)</i>	19
3. Analitzadors sintàctics d'XML	24
3.1. L'API SAX	24
3.1.1. Cicle de vida de l'anàlisi	26
3.1.2. Tractament d'errors	27
3.1.3. Disseny	28
3.1.4. Usos de SAX	31
3.2. L'API DOM	32
3.2.1. La jerarquia d'objectes DOM	33
3.2.2. Tractament d'errors	35
3.2.3. Disseny	35
3.2.4. Usos de DOM	37
3.2.5. JDOM	37
4. Processadors XSLT	39
4.1. Disseny	40
4.2. Exemple d'ús d'un processador XSLT	41
5. L'API JAXP	44
6. Usos i aplicacions de la tecnologia XML	46
6.1. Exemple 1. Aplicacions web	46
6.2. Exemple 2. Rendiment	47
6.3. Exemple 3. Internacionalització (i18n)	48

Resum	49
Activitats	51
Exercicis d'autoavaluació	51
Solucionari	54
Annexos.....	60
Glossari	75
Bibliografia	79

Introducció

En aquest mòdul parlarem dels llenguatges de marques. Un llenguatge de marques és una descripció de la manera d'introduir indicacions especials dins d'un document. Es tracta de marcar el text del document amb instruccions anomenades *marques* o *tags* amb l'objectiu que un ordinador les pugui interpretar.

Veurem que des del 1986 hi ha un estàndard per al marcatge de documents i el relacionarem amb mots com ara *HTML*, *XML*, *DTD*, *XSL*, *XSLT*, etc. Tot això ens servirà d'introducció al que realment serà la part important del mòdul, els **analitzadors sintàctics XML** i els **processadors XSLT**. Veurem què són aquestes eines i l'arquitectura i les funcionalitats que ofereixen, com es relacionen amb els conceptes de compiladors i els importants esforços d'estandardització que s'han fet entorn seu.

Pel que fa als aspectes pràctics, utilitzarem Java com a llenguatge de base. Tant els analitzadors XML com els traductors XSLT poden estar implementats i poden ser usats en molts llenguatges de programació diferents, però nosaltres ens centrarem en el Java. Això no és gratuït, bona part del programari disponible per al tractament XML s'ha desenvolupat en Java: en part perquè molt d'aquest codi s'ha fet públic mitjançant iniciatives de programari lliure i Java és la millor opció, tenint en compte el seu caràcter de llenguatge multiplataforma; en part també, perquè molt d'aquest programari s'utilitza en aplicacions web i Java és, ara per ara, el llenguatge per excel·lència per a crear aplicacions d'aquest tipus. De tota manera, i atès que les tecnologies XML cada cop s'apliquen a més àmbits, els llenguatges de programació en què podem trobar implementacions d'analitzadors sintàctics XML i processadors XSLT és cada cop més gran.

Al final del mòdul hi trobareu un apartat dedicat als usos i les aplicacions actuals de les tecnologies XML i als annexos, hi trobareu informació de referència que us pot ser d'utilitat a l'hora de fer exercicis pràctics o fins i tot per a usos extraacadèmics. L'últim annex, el mapa d'estàndards, pretén que us pugueu situar ràpidament dins del mar de sigles en què indefectiblement es navega quan es parla de tecnologia XML.

Objectius

En acabar l'estudi d'aquest mòdul didàctic, l'estudiant hauria d'haver assolit els objectius següents:

- 1.** Conèixer què són i d'on vénen els llenguatges de marques.
- 2.** Treballar de manera bàsica amb documents XML, XSLT i DTD.
- 3.** Saber quines són les fonts de documentació per a poder buscar més informació i profundir en aquestes tecnologies.
- 4.** Comprendre i utilitzar els analitzadors sintàctics XML. Destriar quan cal utilitzar SAX i quan cal utilitzar DOM.
- 5.** Comprendre i utilitzar els processadors XSLT.
- 6.** Relacionar tant els analitzadors sintàctics XML com els processadors XSLT amb els conceptes de compiladors i saber com s'han de construir.
- 7.** Adonar-se de la importància i l'impacte dels estàndards i de les organitzacions d'estandardització de programari i de formats de dades.

1. Documents electrònics

No és fàcil trobar una definició adequada de document. La definició del diccionari fa referència a informació, cosa escrita, gravada, etc.

Document. Escrit original o oficial que serveix de base, de prova o de suport d'alguna cosa./ Unitat d'informació corresponent a un contingut singular.

Un document és un objecte abstracte que és més fàcil d'imaginar si es pensa en exemples concrets com ara un llibre, un manual, un catàleg, un tríptic, un formulari, un correu electrònic, etc.

Històricament, el medi de presentació d'un document havia estat el paper, però amb l'arribada de l'ordinador va néixer un nou medi: la pantalla electrònica. Els documents que un ordinador pot emmagatzemar, manipular i presentar (per impressora, pantalla o qualsevol altre dispositiu) s'anomenen *documents electrònics*. 

Per a entendre bé què és un document electrònic cal tenir clars els conceptes *reproducció* i *presentació*.

S'anomena **reproducció** el format d'arxiu que conté les dades del document combinades amb la descripció del format* de text que es vol aplicar a aquestes dades.

S'anomena **presentació** el resultat de convertir una *reproducció* en qualcom perceptible per a l'ésser humà.

Document electrònic

Fitxer que un ordinador pot emmagatzemar, manipular i presentar.

* Tot allò que té a veure amb la manera de presentar un text determinat. Per exemple, la font de lletra que s'utilitza (cursiva, negreta, etc.), la sagnia del paràgraf, etc.

Alguns exemples de formats coneguts de reproducció són els següents:

- *TROFF (text run off)*. Aquest format va néixer el 1973 dins del context dels sistemes Unix i és la base de formats molt utilitzats com ara el de la conejuda eina d'ajuda *man* dels sistemes Unix i Linux.
- *RTF (rich text format)*. Aquest format, encara és molt vigent. Molts processadors de textos permeten d'exportar els seus documents en aquest format i altres eines, com ara alguns paquets de programari de traducció lingüística, l'utilitzen com a format d'entrada.
- *LaTeX*. Aquest format té una gran acceptació dins de la comunitat científica, sobretot per la seva gran productivitat a l'hora d'escriure fórmules matemàtiques i per la seva gran qualitat de presentació.

1.1. Marcatge de format

El processament de textos és un dels àmbits dels documents electrònics que han estat més estudiats. El processament de textos és una disciplina de la informàtica dedicada a la creació de programari capaç d'automatitzar parts de la creació d'un document i del procés d'edició. 

Els primers resultats importants d'aquesta disciplina foren els sistemes basats en marques de format. Aquests sistemes permetien que, un cop teclejat el document, l'autor pogués descriure (mitjançant marques o instruccions especials) el format volgut per a cada fragment del document. Un cop marcat, l'ordinador imprimia el document presentant-lo segons el format indicat per les marques.

Exemple

Aquest és un petit exemple de com podria ser un {\bf document} escrit utilitzant un editor basat en {\it marques de format}. Conté paraules en negreta i en cursiva.

Als fragments de text entre claus se'ls vol aplicar un format: \it vol dir aplicar cursiva i \bf vol dir aplicar negreta. En imprimir l'exemple ens apareixeria de la manera següent:

Aquest és un petit exemple de com podria ser un **document** escrit utilitzant un editor basat en *marques de format*. Conté paraules en negreta i en cursiva.

El processament de textos va evolucionar cap al que actualment es coneix per WYSIWYG* ('el que veus és el que obtens') on la interfície d'usuari per a la reproducció del document està dissenyada per a tenir l'aspecte d'una presentació. És a dir, que el que veu i crea l'usuari per pantalla coincideix amb el que sortirà imprès per la impressora. Un exemple conegut de processador de textos que segueix aquest paradigma és Microsoft Word.

* What you see is what you get.

Les tècniques de processament de textos són extraordinàriament eficaces quan es pretén dur a terme el procés complet de crear un document, guardar-lo en una representació concreta i imprimir-lo. Però no ho són tant quan es vol fer gestió documental. És a dir, quan es vol cercar parts dels documents, emmagatzemar-les o recuperar-les, partir documents, unir-los, reproduir-los, transformar-los, etc.

En molts d'aquest casos sorgeixen problemes importants deguts principalment al fet que la major part de documents tenen **diferents representacions de la informació** i, el que és pitjor, aquestes representacions descriuen com ha de ser presentat el contingut però no quins són els components lògics que el formen. Sense conèixer l'estructura lògica del document, és molt difícil per a una màquina interpretar-ne el contingut i poder-lo gestionar.

Per a solucionar aquesta mena de problemes va néixer el concepte *marcatge generalitzat*. 

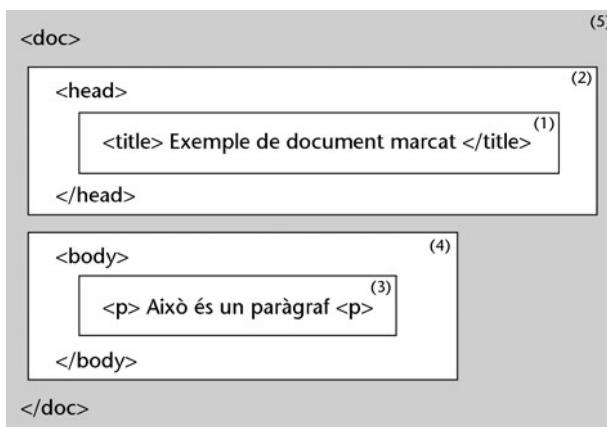
1.2. Marcatge generalitzat

Al final dels anys seixanta IBM volia crear una aplicació per a emmagatzemar, cercar, gestionar i editar documents jurídics. Els tres investigadors encarregats d'avaluar el projecte van conculoure que el principal escull eren els sistemes de

representació dels documents i van proposar crear-ne uns de nous en què les marques identifiquessin l'estructura del document i no el format que s'hi havia d'aplicar.

L'ús de marques que no expressen com cal presentar un element sinó que l'identifiquen, s'anomena *marcatge generalitzat*.

El marcatge generalitzat permet d'identificar blocs d'informació que poden estar constituïts per text o per altres blocs d'informació. El bloc d'informació que engloba tots els altres és el document.



Exemple de blocs d'informació que constitueixen un document. Els blocs d'informació (1), (2), (3), (4) i (5) constitueixen el document. El bloc (2) conté el bloc (1), el bloc (4) conté el bloc (3) i el bloc (5) conté els blocs (2) i (4).

Els documents amb diferent contingut, però constituïts pels mateixos blocs d'informació, són documents d'una mateixa tipologia. El marcatge generalitzat permet de crear models de documents: en escriure un document d'un model determinat, fem una instància d'aquest model de document.

L'any 1969 IBM va desenvolupar el primer llenguatge de marques generalitzat, el GML (*generalized markup language*). Poc després, el principal ideòleg d'aquest llenguatge, Charles F. Goldfarb, va coordinar un grup de l'ISO per tal de crear la Norma ISO 8879, anomenada *llenguatge estandarditzat i generalitzat de marcatge*. L'any 1986 es publicava la versió definitiva d'aquesta norma i neixia l'SGML (*standard generalized markup language*).

GML

Aquesta sigla, a més de ser les inicials del nom del llenguatge, també són les inicials dels noms dels tres investigadors d'IBM que el van crear: Goldfarb, Mosher i Lorie.

L'SGML és, doncs, des del 1986, l'estàndard per a la creació de documents electrònics. Però aquest llenguatge presenta un problema, la seva complexitat. El document que defineix l'estàndard té aproximadament unes cinc-centes pàgines. Les empreses havien de dedicar molt de temps i diners per a crear aplicacions que treballassin amb SGML i no acabaven de sortir eines de propòsit general (editors, analitzadors sintàctics, etc.) per al seu tractament.

Tot i això, l'SGML va influenciar molt el món dels documents electrònics: llenguatges com HTML i CML (per a la notació química) es basen en SGML. L'HTML

(*hypertext markup language*) fou creat el 1989 per Tim Berners-Lee, un investigador del Laboratori Europeu de Partícules Físiques. Berners-Lee volia crear un llenguatge per a publicar treballs d'investigació a la Xarxa, de manera que, mitjançant hipervincles, fos possible visualitzar els seus treballs i els dels seus col·legues. Berners-Lee va crear un tipus de document SGML específic per a aquest propòsit, buscant la màxima simplicitat i amb etiquetes que li permetessin d'indicar jerarquies de títols determinades, negretes, cursives, hiperenllaços i algunes tipologies de paràgraf. Posteriorment, es van construir eines que accedien a la Xarxa, interpretaven aquestes etiquetes i presentaven el document per pantalla. A aquestes eines se'ls va donar el nom de **navegadors web**.

2. L'especificació XML

Als anys noranta, en resposta a la guerra comercial entre els navegadors web Netscape i Explorer, que provocava greus desviacions del llenguatge HTML respecte del seu propòsit original, la comunitat internacional va crear el W3C (World Wide Web Consortium), que havia de vetllar per una estandardització real dels formats de la Xarxa. Una de les primeres tasques del W3C fou oficialitzar un versió unificada d'HTML.

WEB

Com que l'HTML no podia donar resposta a alguns dels usos que s'esperava de la Xarxa (com ara el comerç electrònic, la cerca d'informació, la personalització, etc.), l'any 1998 el W3C va publicar la versió 1.0 de l'especificació XML (*extensible markup language*). L'objectiu d'aquesta especificació és crear un SGML senzill, és a dir, un subconjunt d'SGML que n'elimini les parts més complexes i l'optimitzi per al seu ús a la Xarxa sense deixar de ser, això sí, extensible (el nombre d'etiquetes permeses no és tancat sinó que és possible definir-ne de pròpies). La finalitat de tot plegat és la de crear un llenguatge prou senzill perquè la indústria consideri rendible invertir en la creació d'eines per al seu tractament.

W3C: <http://www.w3.org>

WEB

Podeu trobar l'especificació d'XML en l'adreça següent:
<http://www.w3.org/TR/REC-xml>.
Aquesta especificació té unes vint-i-cinc pàgines, enfront de les cinc-centes que tenia la d'SGML.

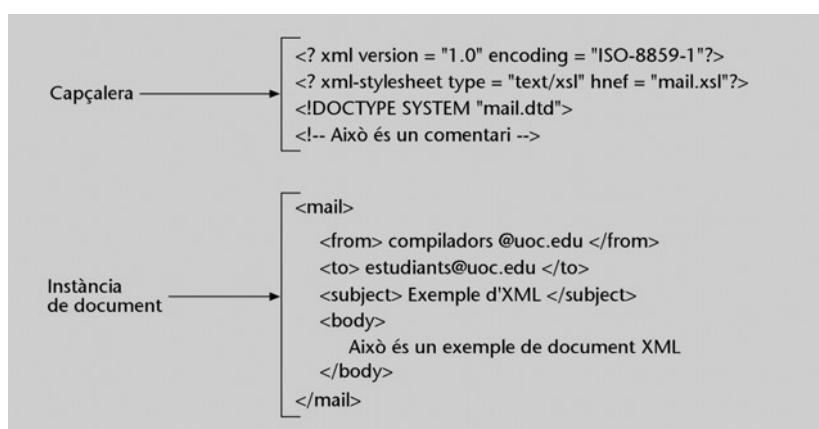
L'XML és un llenguatge de marques generalitzat que pretén dotar els continguts (documents) d'estructura lògica de manera que pugui ser manipulats com a dades per una màquina i es pugui intercanviar informació amb independència de la plataforma, l'arquitectura i la base de dades.

Un document XML és text en una codificació de caràcters concreta. El text XML es pot considerar com un conjunt de **dades de caràcter i marques**. Les *marques* són text que comença pel caràcter < i acaba pel caràcter > i text que comença pel caràcter & i acaba pel caràcter ;. Les *dades de caràcter* són tot allò que no són marques. 

Bibliografia sugerida

Goldfarb, en el seu llibre *The XML Handbook* (referenciat en la bibliografia), ho explica amb un petit joc de paraules: Dades + Marcatge = DocuMent

A més alt nivell, un document XML té dues parts, la **capçalera** i la **instància de document**:



A la **capçalera** hi apareix informació per a la gestió del document, com ara codificació, definició del tipus de document, presentació associada, etc.

La **instància de document** representa el contingut real del document i està constituïda per elements, atributs i dades. Se l'anomena *instància* perquè és una cadena de caràcters que representa un exemplar concret d'un document o d'una tipologia determinada de document.

2.1. Elements

Els elements són les unitats bàsiques que permeten de construir documents. Cada element representa un component lògic del document. Cada un dels blocs d'informació a què fèiem referència en el capítol anterior és un element.

Per a descriure els elements s'utilitzen etiquetes. Una etiqueta consta del **nom del tipus de l'element** i, opcionalment, d'una llista d'atributs que descriuen propietats de l'element.

Per a delimitar l'inici i el final del contingut d'un element s'utilitzen marques (*tags*).

- El principi d'un element es marca de la manera següent:

<nom_tipus_element [atributs]>

Exemple

```
<missatge llenguatge="cat">Hola, bon dia</missatge>
```

- El final d'un element es marca de la manera següent:

</nom_tipus_element>

Els elements buits (sense contingut) es marquen de la manera següent:

<nom_tipus_element/>

Exemple

```

```

El text que hi ha a la marca d'inici, entre els signes < i >, conté la descripció de l'etiqueta de l'element (el nom del tipus de l'element més atributs, si n'hi ha).

El text que hi ha entre la marca d'inici i la de final constitueix el contingut de l'element.

El text que hi ha a la marca de final, entre els signes </ i >, es correspon amb el nom del tipus de l'element.

Quan un element amb contingut no té atributs, les marques d'inici i de final només difereixen en el signe /, que és present a la marca de final però no a la d'inici.

2.2. Atributs

Els atributs descriuen propietats dels elements, representen informació addicional, com ara un identificador o la llengua en què ha estat escrit el contingut de l'element. Els atributs formen part de l'etiqueta de l'element. 

Un atribut es representa mitjançant un parell *nom/valor* separat per un signe =. El valor ha d'anar entre cometes simples o dobles.

Exemple

```

```

`src="computer.gif"` és un atribut que especifica la ubicació, el nom i el format del document que conté la descripció de la imatge.

`alt="Un computador model PC"` descriu en paraules el contingut de la imatge.

2.3. Instruccions de procés

Mitjançant instruccions de procés es poden especificar instruccions per a l'analitzador sintàctic o informació per a les aplicacions. Les instruccions de procés tenen la sintaxi següent:

```
<?instruccio atributs?>
```

Exemple

La instrucció de procés `<?xml version=""1.0" encoding="ISO-8859-1"?>` indica a l'analitzador sintàctic que el document XML s'ha creat seguint l'especificació versió 1.0 d'XML i utilitzant la codificació de caràcters ISO-8859-1.

2.4. Sintaxi

Els documents XML han de complir un seguit de normes sintàctiques:

- Els documents XML han de tenir una estructura d'etiquetes jerarquitzada.

Correcte: <PERSONA><NOM> </NOM></PERSONA>
Incorrecte: <PERSONA><NOM> </PERSONA></NOM>

- Els documents XML han de tenir un element arrel únic.

Correcte: <AMICS>
 <PERSONA><NOM>Pep</NOM></PERSONA>
 <PERSONA><NOM>Pau</NOM></PERSONA>
 </AMICS>

Incorrecte: <PERSONA><NOM>Pep</NOM></PERSONA>
 <PERSONA><NOM>Pau</NOM></PERSONA>

- Una etiqueta buida és una etiqueta que no conté res, és a dir, que és, per si mateixa, inici i final.

Exemple: <PERSONA NOM="Jordi" COGNOM="Pi" />
 <DATA DIA="25" MES="1" ANY="2003" />

- Els valors dels atributs han d'anar entre cometes (simples o dobles).

Exemple: <DATA DIA="25" MES='1' ANY="2003" />

- Es distingeix entre majúscules i minúscules.

Correcte: <nom>Pep</nom>
Incorrecte: <nom>Pep</NOM>

- Els espais en blanc són irrellevants (s'eliminen o es normalitzen).

Les expressions següents són equivalents:

<nom> Pep </nom>
<nom>Pep</nom>

- Cal codificar els caràcters reservats

Taula de conversions	Caràcter reservat	Codificació
	<	<
	&	&
	>	>
	"	"

- L'XML permet d'utilitzar diferents formats de codificació de caràcters.

Exemples

```
<?xml encoding="UTF-8"?>
<?xml encoding="ISO-8859-1"?>
```

Un document és un document XML **ben format** si s'ajusta a la sintaxi del llenguatge XML. Cap document que no estigui ben format és un document XML.

Codificació

És important escollir la codificació apropiada per a no provocar efectes no esperats en els documents. Per a documents en llengües llatines la codificació ISO-8859-1 és la més adequada.

Com que XML, igual que els llenguatges de programació, és un llenguatge que ha de ser entès per la màquina, és bàsic que hi hagi eines que permetin d'anàlitzar els documents. Les màquines, almenys de moment, són poc tolerants amb els errors lèxics i sintàctics, els humans som pràcticament incapços de no cometre'n i, a més, estem totalment preparats per a ignorar-los. Així, doncs, si volem assegurar que el que escrivim sigui entès per les màquines, ara per ara, és imprescindible analitzar-ho mitjançant eines informàtiques.

Un primer procés d'anàlisi que s'aplica sobre un *possible* document XML és comprovar si es tracta d'un document ben format o no. La totalitat de les eines que permeten de manipular documents XML fan aquesta funció i la gran majoria, en cas de trobar-se amb un document que no sigui un document XML ben format, informen dels errors trobats i de la seva posició dins del document.

2.5. L'arbre XML

Un document XML s'ha de poder representar en forma d'arbre. De fet, no és gratuït que hi hagi restriccions sintàctiques com la que dictamina que un document XML ha de tenir un element que englobi tots els altres. Aquesta regla està enfocada a forçar que qualsevol document XML pugui ser representat en forma d'arbre amb una única arrel.

Els analitzadors sintàctics XML, i en general quasi totes les eines informàtiques que tracten amb documents XML, construeixen una representació en forma d'arbre del document. 

Un arbre XML es construeix de la manera següent:

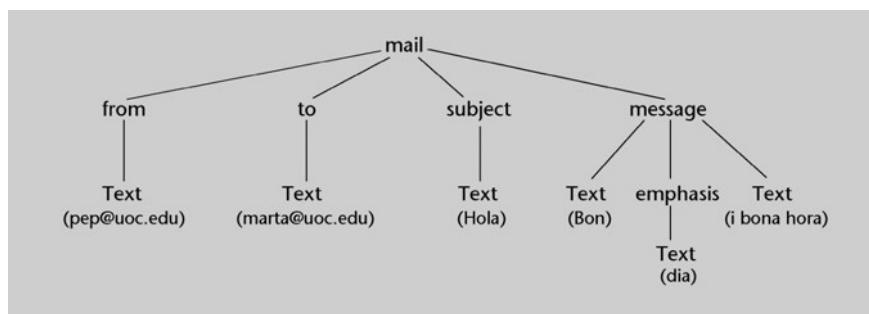
- Els elements del document XML es corresponen amb els nodes de l'arbre.
- El node que representa un element *A* és pare del node que representa un element *B*, si *B* està contingut en *A*.
- Els nodes germans estan ordenats segons l'ordre d'aparició, en el document, dels elements que representen.
- Les dades de caràcter del document XML es consideren nodes de l'arbre etiquetats amb el nom ***Text***.
- Els atributs es consideren informació associada a cada node de l'arbre i, per tant, no apareixen en la representació en forma d'arbre.

Exemple

Considerem el document XML següent:

```
<mail org="UOC">
  <from>pep@uoc.edu</from>
  <to>marta@uoc.edu</to>
  <subject>Hola.</subject>
  <message>Bon <emphasis>dia</emphasis> i bona hora.</message>
</mail>
```

La seva representació en forma d'arbre seria la següent:



A vegades va bé anotar entre parèntesis en els nodes ***Text*** el text a què fan referència, tot i que formalment aquesta informació no hauria d'aparèixer en l'arbre. Nosaltres ho hem fet per facilitar la comprensió de l'exemple.

A partir d'ara, quan parlem d'arbre XML ens referirem a una estructura com la presentada en l'exemple anterior. 

2.6. Treballar amb XML

Per a treballar amb XML cal tenir present un conjunt de conceptes, especificacions i tecnologies molt relacionades.

En una primera aproximació, s'identifiquen tres tipus de documents que cal crear sempre, o gairebé sempre:

- 1) **El DTD (*document type definition*) o esquema (*schema*):** representació formal de les restriccions que han de satisfer els XML quant a estructura i elements. La seva funció principal és permetre de validar de manera automàtica si un document XML compleix aquestes restriccions o no.
- 2) **Els XML:** contenen informació estructurada segons els criteris que marca el DTD.
- 3) **Els XSL:** defineixen la *presentació* que es vol donar a un document XML.

De seguida parlarem més a fons dels documents DTD i dels documents XSL, però abans introduirem un concepte molt útil a l'hora de treballar amb documents XML, els **espais de noms**.

2.6.1. Espai de noms

Una organització fictícia X fa una proposta d'estàndard XML per a representar informació sobre llibres.

Un XML d'exemple que respondria a aquest estàndard podria ser el següent:

```
<llibre>
  <titol>Noves Tècniques de Compiladors</titol>
  <ISBN>00-000-0000-0</ISBN>
</llibre>
```

Una segona organització fictícia Y treballa també en una proposta d'estàndard XML per a representar informació sobre autors.

Un XML d'exemple corresponent a aquesta proposta podria ser el següent:

```
</autor>
  <nom>Pere Token Yacc</nom>
  <titol>Enginyer en Informàtica</titol>
</autor>
```

Un organisme d'entitat superior llegeix les dues propostes i planteja als dos organismes, X i Y, de treballar conjuntament per fer una proposta d'estàndard que permeti de representar informació sobre llibres ampliada amb informació sobre els autors.

Els dos organismes accepten de bon grat i es fa una reunió. Decideixen simplement incloure la informació de l'autor dins de l'etiqueta llibre:

```
<llibre>
  <titol>Noves Tècniques de Compiladors</titol>
  <ISBN>00-000-0000-0</ISBN>
```

```
<nom>Pere Token Yacc</nom>
<titol>Enginyer en Informàtica</titol>
</llibre>
```

Quan s'adonen que el document és ambigu perquè han repetit el nom de l'etiqueta **titol**, s'estableix un debat sobre quina de les dues etiquetes **titol** hauria de canviar de nom. Sobre aquest punt no es posen gaire d'accord perquè els dos organismes reclamen la paternitat de l'etiqueta.

L'XML ofereix un mecanisme per tal de resoldre aquests tipus d'ambigüïtats que pot acontentar els dos organismes de l'exemple. Es tracta de conservar intacte el nom de l'etiqueta però afegint-hi al davant un prefix que descrigui l'àmbit de què prové cada una. De cada un d'aquests àmbits se'n diu un **espai de noms (namespace)**.

La nomenclatura que s'utilitza per a definir una etiqueta que pertany a un espai de noms és: `xxx:yyy`. On `xxx` és el prefix que identifica l'espai de noms i `yyy` és el nom de l'etiqueta.

Es pot afegir una definició d'espai de noms dins de qualsevol element XML; de fet, un espai de noms s'especifica mitjançant un atribut i té vigència per a tots els elements continguts dins de l'element que el declara. 

Un espai de noms es declara mitjançant un atribut que té la sintaxi següent:

```
xmlns:prefix = "uri_espai_de_noms".
```

On `prefix` és una paraula que identifica l'espai de noms i `uri_espai_de_noms` és un **URI (uniform resource identifier)** que fa referència a l'espai de noms.

Si l'espai de noms no té prefix (`xmlns="uri_espai_de_noms"`), parlem d'**espai de noms per defecte** i afecta aquells elements que no tenen cap declaració explícita d'espai de noms ni cap prefix associat.

Un **URI** és un identificador universal de recursos. Es tracta d'un seguit de normes per a especificar una referència unívoca a un recurs. Normalment consisteix en una forma ampliada de les URL (localitzador uniforme de recursos o **uniform resource locator**) que s'utilitzen al Web.

Exemple

```
http://www.uoc.edu/compiladores/recurs1
http://www.uoc.edu/compiladores/recurs1#subrecurs1
```

La sintaxi d'un URI és: `protocol://hostname/path[#fragment]`.

L'exemple amb què hem obert l'apartat, un cop lliure d'ambigüïtats (amb la declaració de l'espai de noms i els prefixos corresponents), quedaria així:

```
<llibre xmlns:infoLlibre="uri_esplainoms_llibres"
         xmlns:infoAutors="uri_esplainoms_autors">
  <infoLlibre:titol>
    Noves Tècniques de Compiladors
  </infoLlibre:titol>
  <infoLlibre:ISBN>00-000-0000-0</infoLlibre:ISBN>
```

```
<infoAutors:nom>Pere Token Yacc</infoAutors:nom>
<infoAutors:titol>Enginyer en Informàtica</infoAutors:titol>
</llibre>
```

Un espai de noms és una correspondència entre un prefix i un URI. És un mecanisme especialment dissenyat per a evitar ambigüïtats que no puguin ser resoltes per l'analitzador.

2.6.2. Document type definition (DTD)

Una tipologia de documents és una classe que engloba documents que són semblants. El tipus d'elements que componen el document i la relació entre si són criteris que permeten de classificar documents en tipologies.

Un DTD és el conjunt de normes que s'han de seguir per a crear una representació d'un document XML d'un tipus determinat. Aquestes normes s'expressen mitjançant una notació anomenada **declaració de marcatge**.

La declaració de marcatge és una notació descrita en l'especificació XML. És una manera d'expressar un conjunt de regles gramaticals que permeten d'indicar el següent:

- Els elements i atributs vàlids dins d'un document XML.
- Els elements que es poden utilitzar dins d'altres elements.
- Els elements i atributs opcionals.

Els documents DTD contenen declaracions de marcatge.

Exemple

DTD	Possible XML	
<pre><!ELEMENT note (to, from, heading, body)> <!ELEMENT to (#PCDATA)> <!ELEMENT from (#PCDATA)> <!ELEMENT heading (#PCDATA)> <!ELEMENT body (#PCDATA)></pre>	<pre><note> <to>Estudiants de Compiladors II</to> <from>Pere Token Yacc</from> <heading>Salutacions</heading> <body>Salutacions cordials!</body> </note></pre>	 <p>En l'“Annex 1” hi podeu trobar una petita guia sobre la sintaxi de les declaracions de marcatge.</p>

La notació de les declaracions de marcatge dels documents DTD persegueix dos objectius principals:

- Poder ser creada i llegida amb facilitat per una persona.
- Poder ser interpretada per una màquina. D'aquesta manera, es poden crear eines que validin automàticament si un XML obedeix a una determinada DTD o no.

La majoria d'eines de tractament de documents XML ofereixen funcionalitats per validar si un document XML compleix les regles del seu document DTD associat o no.

Un document XML és **vàlid** si el seu contingut respecta les regles del seu document DTD associat.

És important no barrejar els conceptes de document **vàlid** i de document ben format. Un document XML ha de ser sempre ben format; en canvi, només en podem comprovar la validesa si té un document DTD associat. En cas de no tenir-ne, no podem dir res sobre la seva validesa. 

Els DTD tenen bàsicament dos problemes:

- S'especifiquen en un llenguatge totalment diferent d'XML.
- Tenen molt poca riquesa per a expressar tipus de dades.

Per a solucionar aquests dos problemes es va generar l'especificació *XML schema*.

L'especificació *XML schema* és una notació per a escriure els DTD alternatius a les declaracions de marcatge i aporta bàsicament tres avantatges:

WEB

Podeu trobar l'especificació *XML schema* en la web del W3C: <http://www.w3.org>. En l'adreça <http://www.w3schools.com> hi trobareu manuals i exemples pràctics d'esquemes XML.

- **És notació XML:** això fa que les mateixes eines que s'utilitzen per a analitzar XML es puguin utilitzar per a analitzar esquemes (*schema*) XML.
- **Suporta diferents tipus de dades:** ofereix un conjunt de tipus de dades ja definits com ara tipus de calendari, de moneda, numèrics, etc. Això permet una validació molt més eficaç dels documents XML.
- **És extensible:** permet noves definicions de tipus de dades.

Els esquemes XML són molt millors per a la indústria del programari que no pas les declaracions de marca, però la notació de les declaracions de marca s'escriu amb menys línies que un esquema XML i és molt més semblant a les notacions gramaticals que heu vist al llarg de l'enginyeria. Per tant, en aquesta assignatura treballarem amb declaracions de marcatge. 

2.6.3. Extensible stylesheet language (XSL)

XSL és una especificació del W3C que té com a objectiu proporcionar un conjunt de regles i elements que permeten de definir diferents *presentacions* per als documents XML.

WEB

Podeu trobar informació referent a les especificacions XSL a la web del W3C: <http://www.w3.org>. En l'adreça <http://www.w3schools.com> hi trobareu manuals i exemples pràctics sobre aquestes especificacions.

La creació d'aquest estàndard no va ser una tasca simple: poc abans de la seva publicació definitiva, es va dividir en tres. Es van publicar tres documents

que obeïen a tres especificacions diferents (XSL *formatting objects*, XSL *transformations* i XPath), que conjuntament formen el que entenem per l'*especificació XSL*.

Des de la seva aparició, el terme *XSL* ha estat utilitzat per a definir coses diferents:

- Formalment *XSL* és el nom que rep l'especificació XSL-FO.
- Normalment, els desenvolupadors parlar d'*XSL* per a fer referència a un full d'estil escrit segons l'especificació XSLT.
- Per raons històriques, s'utilitza *XSL* per a fer referència a les tres especificacions: XSLT, XSL-FO i XPath.

Al llarg d'aquest mòdul seguirem la convenció següent: 

- Quan parlem d'**XSL-FO** farem referència a l'especificació XSL *formatting objects*.
- Quan parlem d'**XSLT** farem referència a l'especificació XSL *transformations*.
- Quan parlem d'**XSL** farem referència al conjunt de les tres especificacions: XSL-FO, XSLT i XPath.

XSL-FO

L'estàndard XSL-FO defineix un llenguatge per donar format a la informació. És un estàndard basat en XML que té un conjunt predefinit de cinquanta-set elements, que també s'anomenen *objectes de format* (*formatting objects*) perquè cada un especifica una característica de format del document, i dos-cents quaranta-vuit atributs.

WEB

Podeu trobar tota la informació referent a l'estàndard XSL-FO a la web del W3C: <http://www.w3.org>. En l'adreça <http://www.w3schools.com> hi trobareu manuals i exemples pràctics d'aquest estàndard.

Amb aquest estàndard es pretén que apareguin les eines o els navegadors web que n'interpretin les marques i generin la *presentació* corresponent. XSL-FO permet de definir les diferents àrees que componen una pàgina i ubicar informació en cada una d'elles.

XSL-FO és un format com RTF o LaTeX, amb la diferència que està basat en XML i ha estat creat pel W3C. 

L'estudi d'aquest estàndard queda fora dels objectius d'aquesta assignatura.

Exemple

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <!-- Definició dels tipus de pàgines del document -->
    <fo:layout-master-set>
        <fo:simple-page-master master-name="una_regio">
            <!-- Definició de les regions de què consta la pàgina -->
            <fo:region-body/>
        </fo:simple-page-master>
    </fo:layout-master-set>
    <!-- Definició dels continguts del document -->
    <fo:page-sequence master-name="una_regio">
        <fo:flow flow-name="xsl-region-body">
            <fo:block font-size="20pt" font-family="serif"
line-height="30pt">
                Hello World!
            </fo:block>
        </fo:flow>
    </fo:page-sequence>
</fo:root>
```

XSLT

La *T* d'XSLT correspon a *transformations*; per tant, l'XSLT és un **llenguatge de transformació**. És a dir, un llenguatge que permet d'especificar instruccions que descriuen com cal fer una traducció.

El concepte traductor s'ha estudiat en l'assignatura *Compiladors I*.

Mitjançant instruccions XSLT, un document XML pot ser transformat en un altre XML, en text, en HTML, en XSL-FO o, en general, en qualsevol altre format de sortida textual.

L'XSLT és un llenguatge basat en XML que permet de definir un conjunt de regles per transformar els elements d'un document XML.

WEB

Podeu trobar tota la informació referent a l'estàndard XSLT a la web del W3C: <http://www.w3.org>. En l'adreça <http://www.w3schools.com> hi trobareu manuals i exemples pràctics d'aquest estàndard.

A grans trets, el procés de transformació funciona de la manera següent: es recorre el document XML original i per a cada un dels elements que el formen, es mira si hi ha definida alguna transformació en el document XSLT. Si hi és, s'aplica; si no, es reprèn el procés. Les transformacions s'escriuen en un document de sortida que es retorna com a resultat quan el procés acaba. Les eines que implementen aquest algorisme de transformació s'anomenen **processadors XSLT**.

Els processadors XSLT s'expliquen en el capítol 4 d'aquest mateix mòdul didàctic.

Exemple

El document XML es combina amb el document XSLT i es genera un document HTML com a resultat.

Document XML

```
<?xml version="1.0"?>
<doc>
  <message>Hello World!</message>
  <author>Francesc</author>
</doc>
```

Document XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" omit-xml-declaration="yes"/>
  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="doc/missatge"/></title>
      </head>
      <body>
        <h1><xsl:value-of select="doc/missatge"/></h1>
        <p><xsl:value-of select="doc/autor"/></p>
      </body>
    </html>
  <xsl:template>
</xsl:stylesheet>
```

Resultat (HTML)

```
<html>
  <head>
    <title> Hello World!</title>
  </head>
  <body>
    <h1> Hello World!</h1>
    <p> FRANCES&Ccedil;</p>
  </body>
</html>
```

En l'annex 3 hi podeu trobar una petita guia sobre el llenguatge de marcatge XSLT.

En el capítol 4 d'aquest mateix mòdul didàctic s'explica la manera d'utilitzar les eines que permeten de fer transformacions com la de l'exemple presentat.

XPath

XPath és l'especificació responsable de la sintaxi d'accés i consulta als elements d'un document XML. Es tracta d'un llenguatge d'especificació d'expressions que permet de definir la manera de localitzar un element específic dins d'un document XML. És, en definitiva, un llenguatge per a la cerca i la consulta de la informació emmagatzemada dins d'una estructura XML.

WEB

Podeu trobar tota la documentació referent a l'especificació XPath a la web del W3C: <http://www.w3.org>. En l'adreça <http://www.w3schools.com> trobareu manuals i exemples pràctics d'aquest estàndard.

XPath, a diferència de les dues especificacions que acabem de veure, **no** és un llenguatge basat en XML. 

XPath és un llenguatge d'especificació d'expressions no basat en XML que permet la cerca i la consulta de la informació emmagatzemada dins una estructura XML.

Exemple

Expressió Xpath: `llibre/capitol/titol`

Descripció: accedeix a tots els elements `titol` que estiguin continguts en un element `capitol` que a la vegada estigui contingut en un element `llibre`.

En l'annex 2 hi podeu trobar una petita guia sobre la sintaxi de les expressions XPath.

L'especificació XSLT utilitza l'especificació XPath. Ja hem dit que, bàsicament, un document XSLT consisteix en un conjunt de regles que defineixen transformacions sobre elements d'un document XML. Doncs bé, cada una de les regles té associada una expressió XPath que determina quins són els elements de l'XML als quals s'ha d'aplicar.

Exemple

```
Element XSLT: <xsl:value-of  
select="llibre/capitol/titol[position()=1]" />
```

En aquest exemple, l'expressió XPath apareix en negreta.

Descripció: escriu en el document de sortida el valor del primer element títol de cada capítol d'un llibre.

Una de les possibles transformacions que es pot generar mitjançant XSLT és un document XSL-FO. És a dir, es pot definir un document XSLT que transformi un document XML en un document XSL-FO.

Exemple

El document XML es combina amb el document XSL i es genera un document XSL-FO com a resultat.

```
XML  
<?xml version="1.0"?>  
<doc>  
  <missatge>Hello World!</missatge>  
  <autor>Francesc</autor>  
</doc>
```

Podeu trobar FOP, una eina que permet de generar documents PDF a partir de documents XSL-FO en l'adreça següent: <http://www.apache.org>.

WEB

XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:output method="xml"/>
<xsl:template match="/">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <fo:simple-page-master master-name="a_region">
        <fo:region-body/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-name="a_region">
      <fo:flow flow-name="xsl-region-body">
        <fo:block font-size="40pt" font-family="serif" line-height="30pt">
          <xsl:value-of select="doc/message"/>
        </fo:block>
        <fo:block font-size="20pt" font-family="serif" line-height="30pt">
          <xsl:value-of select="doc/author"/>
        </fo:block>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>
</xsl:stylesheet>
```

Tornareu a trobar aquest exemple
en el capítol "Processadors XSLT"
d'aquest mateix mòdul didàctic.

**Resultat (XSL-FO)**

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="a_region">
      <fo:region-body/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-name="a_region">
    <fo:flow flow-name="xsl-region-body">
      <fo:block line-height="30pt" font-family="serif" font-size="40pt">Hello World!</fo:block>
      <fo:block line-height="30pt" font-family="serif" font-size="20pt">Francesc</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

3. Analitzadors sintàctics d'XML

Un analitzador sintàctic d'XML és un programa que llegeix dades XML d'una font d'entrada i en fa l'anàlisi lèxica i sintàctica. Si durant el procés d'anàlisi es produueixen errors, aquests programes han de retornar missatges que indiquen l'error detectat i el lloc on s'ha produït.

La majoria d'analitzadors sintàctics XML tenen dos modes de funcionament:

- **Sense validació:** només comprova si el document està **ben format**,
- **Amb validació:** comprova tant si el document està **ben format** com si és **vàlid**.

Els analitzadors sintàctics XML estan pensats com a components de processament de dades que poden ser integrats dins dels programes, no tenen gaire sentit si no se'ls connecta a altres components que facin alguna cosa amb les dades processades.

Recordeu que dins de l'arquitectura general d'un compilador, el mòdul d'anàlisi sintàctica només en constitueix una part. De manera similar, un analitzador sintàctic XML, malgrat que normalment no s'utilitzi per a construir compiladors, és part d'un programa més gran.

Això fa que sigui molt important disposar d'una API* (interfície per a programar l'aplicació) que permeti de connectar fàcilment l'analitzador amb la resta del programa.

* Application programming interface.

Hi ha dues definicions d'API que segueixen, o haurien de seguir, tots els fabricants d'analitzadors sintàctics XML: API SAX i API DOM. Aquestes dues API tenen propòsits i orígens diferents, però amb aquestes es pot fer qualsevol tipus de tractament sobre documents XML.

Per qüestions d'eficiència és molt important saber identificar quina és l'API més adequada per a cada tipus de problema. Equivocar-se en aquesta decisió pot voler dir gastar molta més memòria de la necessària o fer que el temps d'execució sigui molt més elevat del que caldría.

3.1. L'API SAX

SAX és l'acrònim de *simple API for XML* (API simple per a XML). SAX és programari de lliure distribució, totalment gratuït i amb el codi obert, creat a par-

Podeu trobar l'API SAX, i tota la documentació relacionada, en el grup de discussió XML-DEV: <http://www.xml.org/xml-dev>. I, també, a la web de David Megginson: <http://www.megginson.com/SAX>.

WEB

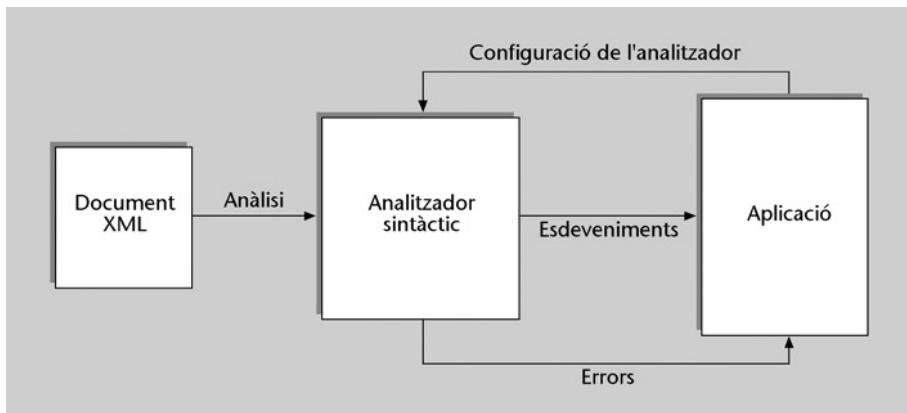
tir de les aportacions fetes per desenvolupadors XML de tot el món al grup de discussió **XML-DEV**. SAX es troba disponible en força llenguatges de programació (JAVA, C, C++, etc.).

SAX llegeix un document XML de manera seqüencial. Al llarg de tot el procés de lectura, en fa una anàlisi sintàctica i genera un conjunt d'esdeveniments o *events* cada cop que identifica una cosa significativa.

SAX proporciona, a les aplicacions que l'utilitzen, un conjunt d'esdeveniments que informen sobre el contingut i l'estructura del document XML analitzat. En el subapartat següent veurem de quins esdeveniments es tracta i com es poden capturar.

Quan una aplicació vol usar SAX, primer el configura indicant-li, per exemple, quins esdeveniments vol rebre. A partir d'aquest moment SAX, cada cop que analitza un document XML, notifica a l'aplicació els esdeveniments i, en cas d'haver-n'hi, els errors.

La figura següent mostra el model d'un analitzador sintàtic XML basat en SAX:



A partir d'ara, accompanyarem les explicacions amb exemples concrets d'ús de SAX i, per a fer-ho, ens centrarem en la seva versió Java.

En Java, l'API SAX és un conjunt d'interfícies o *interfaces* i classes que defineixen un conjunt de mètodes que poden ser usats o redefinits des de les aplicacions que construïm.

Els exemples que trobeu aquí han estat escrits mitjançant l'API SAX de l'anàlitzador sintàtic XML **Xerces**, el codi del qual, juntament amb la documentació associada, el podeu trobar en l'adreça següent:
<http://www.apache.org>.

En el fitxer de codi **SAXParser1.java** hi trobareu un primer exemple de la manera en què s'ha de construir una aplicació Java que utilitza l'API SAX de l'anàlitzador Xerces. Si proveu aquest programa passant-li un document XML, veureu que aparentment no fa res. Per a fer que SAX tingui un comportament determinat, primer cal configurar-lo implementant algunes interfícies i redefinint alguns mètodes.

El fitxer **SAXParser1.java** el podeu trobar a l'aula de l'assignatura.

WEB

3.1.1. Cicle de vida de l'anàlisi

Per a saber què cal redefinir i com, és important conèixer com analitza SAX un document XML. Aquest procés s'anomena el *cicle de vida de l'anàlisi SAX*.

El conjunt d'esdeveniments que llança SAX durant l'anàlisi sintàctica es troben definits en la interfície `org.xml.sax.ContentHandler`. Tot seguit podeu veure un quadre resum dels més importants:

WEB
El codi de la interfície
`ContentHandler`
(`ContentHandler.java`)
el podeu trobar a l'aula de l'assignatura.

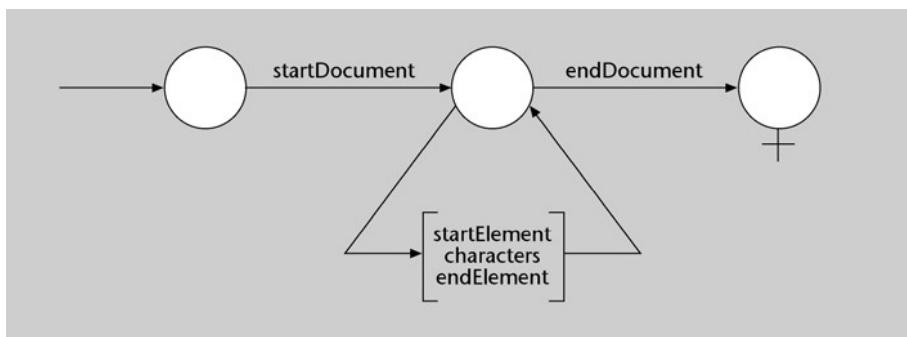
En aquesta taula no apareixen tots els esdeveniments disponibles en la interfície `org.xml.sax.ContentHandler`. Cal que us mireu la documentació de SAX si en voleu tenir una visió exhaustiva.

Esdeveniment	Descripció de l'esdeveniment	Paràmetres de l'esdeveniment	Nota
<code>startDocument</code>	Es produeix en el moment en què comença l'anàlisi del document.	No té paràmetres.	Tots els esdeveniments poden produir una excepció de tipus <code>org.xml.sax.SAXException</code> . Si en tractar algun d'aquests esdeveniments es troba un error, es pot llançar aquesta excepció que notifica el problema detectat i atura el procés d'anàlisi.
<code>startElement</code>	Es produeix just després de tractar el token ' <code>></code> ' de la marca d'inici d'un element XML.	<ul style="list-style-type: none"> - namespaceURI: URI de l'espai de noms associat a l'element. Val <code>null</code> en cas de no tenir-ne. - localName: nom de l'element sense el prefix de l'espai de noms. - rawName: nom complet de l'element en la forma: <code>[namespacePrefix]: [localName]</code>. - attr: objecte que compleix la interfície <code>Attributes</code> de l'API SAX. Aquesta interfície proporciona mètodes per a accedir als noms dels atributs de l'element i als seus valors. 	
<code>endElement</code>	Es produeix just després de tractar el token ' <code>></code> ' de la marca de final d'un element XML.	<ul style="list-style-type: none"> - namespaceURI: URI de l'espai de noms associat a l'element. Val <code>null</code> en cas de no tenir-ne. - localName: nom de l'element sense el prefix de l'espai de noms. - rawName: nom complet de l'element en la forma <code>[namespacePrefix]: [localName]</code>. 	
<code>characters</code>	Es produeix just després d'analitzar les dades de caràcter contingudes en un element XML.	<ul style="list-style-type: none"> - ch: vector de caràcters on cada posició es correspon amb un dels caràcters del bloc de dades analitzat. - start: posició del primer caràcter del vector. - end: posició de l'últim caràcter del vector. <p>Així, doncs, els caràcters que hi ha en el vector ch entre les posicions start i end corresponen als caràcters del bloc de dades analitzat.</p>	<p>En algunes implementacions de SAX, per a analitzar un bloc de dades, es genera més d'un event <code>characters</code>; en canvi, en d'altres, el mateix bloc de dades genera només un <code>event characters</code>. Cal tenir-ho en compte per a recollir correctament aquestes dades. La millor solució és suposar que un mateix bloc de dades pot generar múltiples <code>events characters</code>, així els programes funcionaran tant en els casos que l'anàlitzador generi un sol esdeveniment com en els que en generi múltiples.</p>

Per a poder treballar còmodament amb aquest conjunt d'esdeveniments, l'API SAX ofereix una classe anomenada `DefaultHandler` que implementa la interfície `ContentHandler`. `DefaultHandler` ofereix implementacions buides per a tots els mètodes de la interfície `ContentHandler`. Si es vol utilitzar `DefaultHandler` només cal crear-ne una subclasse i redefinir els mètodes que interessa per a fer el tractament volgut sobre el document XML.

WEB
A l'aula de l'assignatura hi trobareu el fitxer `SAXParser2.java`, que conté un exemple d'ús de SAX que implementa la interfície `DefaultHandler` per tal de configurar el tractament alguns dels `events SAX`.

El diagrama següent del cicle de vida de l'anàlisi amb SAX il·lustra el moment en què es pot produir cada esdeveniment al llarg del procés d'anàlisi d'un document XML:



3.1.2. Tractament d'errors

Pel que fa al tractament d'errors, SAX desencadena esdeveniments que notifiquen errors quan identifica, durant el procés d'anàlisi, alguna circumstància anòmala, com ara un element que no té etiqueta de tancament o el valor d'un atribut que no està entre cometes.

La interfície que proporciona SAX per al tractament d'errors és `ErrorHandler` i el seu funcionament és molt similar al de la interfície `ContentHandler`. `ErrorHandler` defineix tres mètodes, que permeten de personalitzar el tractament d'errors:

WEB
El codi de la interfície
`ErrorHandler`
(`ErrorHandler.java`)
el podeu trobar a l'aula de l'assignatura.

Esdeveniment	Descripció de l'esdeveniment	Paràmetres de l'esdeveniment
<code>warning</code>	Es produeix quan l'analitzador detecta alguna anomalia lleu; no cal que sigui una vulneració d'una regla d'XML, sinó que pot ser una incorrecció o una mancança en el document analitzat.	Un objecte del tipus: <code>SAXParseException</code>
<code>error</code>	Es produeix quan l'analitzador detecta una violació d'alguna regla d'XML que permet, però, de continuar l'anàlisi del document. Normalment es tracta d'errors de validació en el document.	Un objecte del tipus: <code>SAXParseException</code>
<code>fatalError</code>	Es produeix quan l'analitzador detecta una violació d'alguna regla d'XML que fa que l'analitzador hagi de parar el seu procés d'anàlisi.	Un objecte del tipus: <code>SAXParseException</code>

Tots els esdeveniments d'`ErrorHandler` poden produir una excepció de tipus `org.xml.sax.SAXException`.

Tots els mètodes reben com a paràmetre l'excepció `SAXParseException`. Aquest objecte conté el número de línia en què s'ha detectat el problema, l'URI del document i la informació típica de totes les excepcions Java (com ara la descripció de l'error i la traça de la pila de crides a mètodes).

Tots els esdeveniments poden llançar l' excepció SAXException. Tot i que pot semblar una redundància, no ho és si es fa una bona distinció entre el que són els errors que genera l' analitzador **com a conseqüència de l' anàlisi sintàctica del document** i els que genera **com a conseqüència d' errades de l' aplicació** (com ara errors d' entrada/sortida, falta de memòria, etc.). 

A l'aula de l'assignatura hi trobareu el codi de la interfície **(SAXParseException.java)** i el fitxer de codi **SAXParser3.java**. Aquest fitxer conté un exemple d' esdeveniments de **ErrorHandler** que llancen l' excepció **SAXException**.

Els errors que genera l' analitzador com a conseqüència de l' anàlisi sintàctica del document es tracten redefinint la interfície **ErrorHandler**.

Els errors que genera l' analitzador com a conseqüència d' errades de l' aplicació cal notificar-les mitjançant l' excepció **org.xml.sax.SAXException**.

Sempre que es llança una excepció **org.xml.sax.SAXException** des de qualsevol mètode de tractament d' esdeveniments (tant si és un dels definits a **ContentHandler**, a **ErrorHandler** com a qualsevol altra interfície de definició d' esdeveniments) és propagada per l' aplicació i provoca la **interrupció del procés d' anàlisi**.

Resumint, SAX genera tres tipus d' errors:

- **Warnings**, que són més aviat notificacions d' anomalies o situacions estranyes.
- **Errors no fatal**s, que són errors que no impedeixen que l' analitzador pugui continuar endavant.
- **Errors fatal**s, que provoquen que l' anàlisi hagi d' acabar.

Tant els **warnings** com els errors no fatal solen estar associats a errors provocats per documents **no vàlids**, mentre que els errors fatal normalment són errors provocats per documents **mal formats** que no s' ajusten a la sintaxi XML.

3.1.3. Disseny

En aquest apartat veurem com construir un analitzador sintàctic SAX molt bàsic a partir dels conceptes de compiladors que ja s' han estudiat.

Aquest analitzador sintàctic SAX haurà d' analitzar documents XML que s' ajustin a les restriccions següents:

- No hi haurà instruccions de procés. No hi haurà ni tan sols la definició de document XML: `<?xml version="1.0"?>`.
- Treballaran amb codificació ASCII.

- No hi haurà entitats ni caràcters codificats; això vol dir que els documents creats no podran tenir com a dades de caràcter, '<' i '>'.
- No hi haurà comentaris.
- No hi haurà cap definició de DTD ni d'esquema associat.
- No hi haurà definicions d'espais de noms.
- No hi haurà elements CDATA.

En primer lloc, s'ha de determinar quins mòduls calen per a construir un analitzador SAX. En aquest cas, tenint en compte que se'n pretén construir un de molt bàsic, només caldrà un analitzador lèxic i un analitzador sintàctic.

Les tècniques per a programar i combinar cada una de les parts no diferiran gaire de les que ja coneixeu de *Compiladors I*:

- 1) Caldrà definir els lexemes que es poden trobar dins del document XML, com ara "<", ">", "/", els noms dels elements, etc.
- 2) Caldrà crear una gramàtica que reconegui la sintaxi d'XML.
- 3) Caldrà escriure accions en les produccions de la gramàtica que permetin de generar els esdeveniments de SAX.

Per a portar a terme el pas 1 es pot utilitzar un generador d'analitzadors lèxics (com ara LEX). Aquests són els lexemes que caldrà identificar:

Token	Expressió regular (LEX)	Descripció
MES_GRAN	">"	El caràcter més gran.
MES_PETIT	"<"	El caràcter més petit.
NOM	(a-zA-z_)(a-zA-Z0-9_.`-:)*	Un nom (d'etiqueta o d'atribut).
CADENA	\\"[^\"]\\"	Una cadena entre cometes dobles.
CADENA	'[^']'	Una cadena entre cometes simples.
BARRA	"/"	La barra de tancament de marca.
IGUAL	=	Signe d'igual.
PCDATA	[^>\<]/*	Contingut PCDATA d'una etiqueta. S'ha de posar al principi del fitxer de LEX perquè tingui preferència sobre les altres regles.

Per a portar a terme el pas 2, la gramàtica es podria definir de la manera següent:

```
document → elem
elem     → emptyTag
          | startTag   contingut   endTag
```

La gramàtica s'ha descrit utilitzant la notació **BNF** (*backus-naur form visual*). Noteu que els símbols terminals estan en negreta.

```

emptyTag → MES_PETIT NOM atributs BARRA MES_GRAN
startTag → MES_PETIT NOM atributs MES_GRAN
endTag → MES_PETIT BARRA NOM MES_GRAN
atributs → NOM IGUAL CADENA atributs
| ε
contingut → elem contingut
| PCDATA contingut
| ε

```

Per a la implementació d'aquest analitzador SAX en Java, caldria escriure aquesta gramàtica amb un generador d'analitzadors sintàctics. Es tractaria que el generador sintàctic utilitzat generés una classe anomenada SAXParser que implementés la interfície org.xml.sax.XMLReader:

```

Class SAXParser implements org.xml.sax.XMLReader {
    private org.xml.sax.ErrorHandler errorH;
    private org.xml.sax.ContentHandler contentH;
    ...
}

```

 La utilitat dels atributs ErrorH i contentH s'explicarà una mica més endavant.

Caldria, també, redefinir tots els mètodes de la interfície XMLReader. Es deixaran tots buits menys els tres que interessen:

```

public void setErrorHandler(ErrorHandler handler) {
    // a errorH hi guardem la classe que fa el tractament d'errors.
    errorH = handler;
}

public void setContentHandler(ContentHandler handler) {
    // a contentH hi guardem la classe que tracta els esdeveniments de
    // l'anàlisi.
    contentH = handler;
}

public void parse (string systemId) throws IOException, SAXException {
    // aquí hi va el codi que activa l'analitzador sintàtic. */
}

```

Pel que fa al pas 3, el tractament d'errors i la generació dels esdeveniments de l'anàlisi, cal recordar que SAX genera esdeveniments que notifiquen tres tipus d'errors (*warnings*, *errors* i *errors fatal*s). Atès que l'analitzador SAX que es pretén construir no valida, els errors que generarem seran sempre *fatal*s. La manera de llançar-los és crear una SAXParseException, emplenar-la amb la informació que dóna l'analitzador sintàctic o *parser* sobre l'error, el número de línia, etc. i cridar el mètode errorH.fatalError (novaExcepcio). Presentem, ara, un exemple que il·lustra el cas en què l'etiqueta de tancament no coincideix amb la d'obertura:

```

elem → emptyTag
| startTag contingut endTag
{
    if($1!=$3) {
        SAXParseException novaExcepcio;
        novaExcepcio = new SAXParseException(...);
        errorH.fatalError(novaExcepcio);
        System.exit(-1);
    }
}
startTag → MES_PETIT NOM atributs MES_GRAN {$$=$2;}
endTag → MES_PETIT BARRA NOM MES_GRAN {$$=$3;}

```

Notació dels atributs

Els atributs dels símbols de la gramàtica s'han especificat amb la notació de YACC:

\$\$ fa referència a l'atribut del símbol de la part esquerra d'una producció.

\$1, \$2, etc. fan referència als símbols de la part dreta de la producció (\$1 correspon al primer element, \$2 al segon i així successivament).

Un cop tractats els errors, ja només resta generar els esdeveniments corresponents a cada element analitzat.

Caldrà una classe de suport que implementi la interfície Attributes:

```
Class LlistaAtributs extend Vector
implements org.xml.sax.Attributes {
    /* aquesta classe implementa una llista d'atributs mitjançant un
    vector, vosaltres mateixos la podeu completar */
    ...
}
```

Vegem la gramàtica, amb les accions que generen els esdeveniments incloses:

```
document → {contentH.startDocument();} elem
{contentH.endDocument();}

elem → emptyTag
| startTag contingut endTag

emptyTag → MES_PETIT NOM atributs BARRA MES_GRAN
{
    contentH.startElement("", $2, $2, $3);
    contentH.endElement("", $2, $2);
}

startTag → MES_PETIT NOM atributs MES_GRAN
{
    $$=$2;
    contentH.startElement("", $2, $2, $3);
}

endTag → MES_PETIT BARRA NOM MES_GRAN
{
    $$=$3;
    contentH.endElement("", $3, $3);
}

atributs → NOM IGUAL CADENA atributs
{
    $4.addAttribute($1, $3);
    $$= $4;
}
| ε { $$= new LlistaAtributs();}

contingut → elem contingut
| pcdata_aux contingut
| ε

pcdata_aux → PCDATA {contentH.characters($1, 0, $1.length);}
```

Podeu provar de completar aquest codi i introduir-lo en l'eina que utilitzeu per a crear l'anàlitzador sintàctic. També podeu provar d'ampliar la gramàtica amb altres produccions i, fins i tot, podeu provar d'afegeir-hi validació, tot i que això farà força més extensa la gramàtica i afegeix la necessitat d'utilitzar una taula de símbols.

3.1.4. Usos de SAX

SAX és molt útil per a processos de cerca d'elements o d'accés a informació específica dins del document XML. És força ràpid i no gasta memòria; a més, quan es localitza l'element cercat, permet de parar el procés d'anàlisi.

SAX no permet un accés aleatori a un document XML, sinó que llegeix el document d'una manera seqüencial i va llançant esdeveniments mentre dura aquest procés de lectura: si un cop llegit el document es volgués tornar a accedir a la informació d'un element determinat, no hi hauria cap més remei que tornar a analitzar el document des del principi fins a trobar l'element en qüestió.

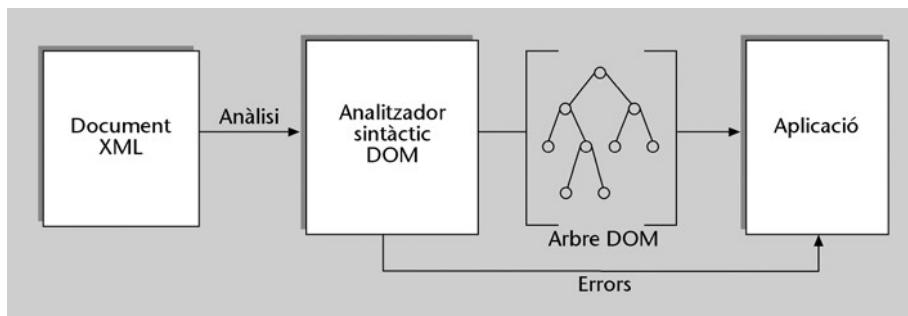
3.2. L'API DOM

Per a poder accedir ràpidament i tantes vegades com es vulgui a la informació dels elements del document XML, caldria construir-ne una representació en memòria. Això és precisament el que fa un analitzador DOM, construeix una representació en memòria d'un document XML. DOM és la sigla de *document object model* ('model d'objectes del document') i és una especificació de l'organisme internacional W3C.

WEB

Podeu trobar tota la informació referent a l'especificació DOM (*document object model*) a: <http://www.w3.org>.

DOM defineix una estructura d'arbre amb mètodes i propietats per a poder tant navegar per l'estructura com obtenir-ne les dades emmagatzemades.



Model d'un analitzador sintàctic XML basat en DOM.

Els analitzadors sintàctics basats en DOM retornen un arbre DOM a les aplicacions que els utilitzen. En cas de produir-se errors durant l'anàlisi, també n'informen a l'aplicació.

L'API DOM està especificada en CORBA IDF (*common object request broker architecture interface definition language*) i, per tant, és una especificació independent de la plataforma i del llenguatge de programació.

L'especificació DOM no s'organitza per versions sinó per nivells:

- **DOM nivell 1:** (<http://www.w3.org/TR/REC-DOM-Level-1>). Detalla la funcionalitat i la navegació que ha de tenir una representació en memòria d'un document. Entenen *document* com una classe abstracta de document electrònic.
- **DOM nivell 2:** (<http://www.w3.org/TR/REC-DOM-Level-2>). Detalla funcionalitats i diferents maneres de representar en memòria tipologies de documents, com ara documents XML, documents HTML o documents CSS.
- **DOM nivell 3:** (<http://www.w3.org/TR/REC-DOM-Level-3>). Detalla tot allò que té a veure amb el procés de generació del DOM, com ara habilitar la validació de document, especificar els mètodes que hauria de tenir un

Nota

La majoria d'analitzadors actuals segueixen les especificacions de nivell 1 i nivell 2 força fidelment. Pel que fa al nivell 3, és recomanable mirar les especificacions de cada producte per a veure quina part cobreixen.

analitzador sintàctic de DOM, quins errors es poden produir durant el procés d'anàlisi i com s'han de notificar, etc.

Utilitzar DOM per a un llenguatge de programació concret requereix disposar d'un conjunt de classes, escrites en aquest llenguatge, que implementin les especificacions DOM. Aquests desenvolupaments constitueixen les API, pròpies de cada llenguatge, per a crear i manipular estructures DOM.

L'explicació de l'analitzador sintàctic basat en DOM s'acompanyarà amb exemples d'ús concrets. Tal com s'ha fet en explicar SAX, s'utilitzarà Java com a llenguatge de programació i el Xerces com a analitzador (Xerces permet de fer anàlisis tant SAX com DOM).

WEB

Podeu trobar l'analitzador sintàctic Xerces, la documentació associada i el seu codi en Java, a: <http://www.apache.org>.

WEB

El fitxer `DOMParser1.java` és un primer exemple de la manera en què s'ha de construir una aplicació Java que utilitza l'API DOM de l'analitzador Xerces, i el trobareu en l'aula de l'assignatura.

3.2.1. La jerarquia d'objectes DOM

En aquest apartat s'explica, a partir d'un exemple, la jerarquia d'objectes que ofereix DOM i la manera de representar un document XML en DOM.

Suposem que disposem de l'XML següent:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xmlstylesheet type="text/xsl" href="llibre.xsl"?>
<!DOCTYPE catalog SYSTEM "llibre.dtd">
<cataleg>
    <llibre id="L1">
        <autor>Pere Token Yacc</autor>
        <titol>Compiladors i XML</titol>
        <preu>20,50</preu>
        <data>01-01-2003</data>
        <descripcio idioma="cat">La influència de les tècniques de
            compiladors en les eines de tractament XML</descripcio>
    </llibre>
</cataleg>
```

La representació en memòria mitjançant objectes DOM del document seria la següent:

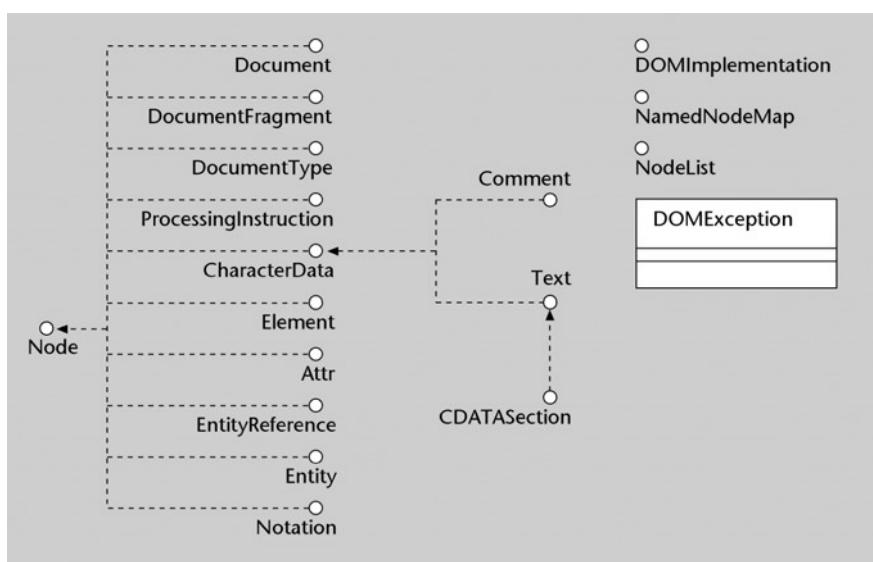
```
Document (tot el document)
└─ ProcessingInstruction (declaració XML)
└─ ProcessingInstruction (XML-stylesheet)
└─ DocumentType (DOCTYPE)
└─ Element (cataleg)
    └─ Element (llibre)
        └─ NamedNodeMap
            └─ Attr (id="L1")
        └─ Element (autor)
            └─ Text (Pere Token Yacc)
        └─ Element (titol)
            └─ Text (Compiladors...)
        └─ Element (preu)
            └─ Text (20,50)
        └─ Element (data)
            └─ Text (01-01-2003)
        └─ Element (descripcio)
            └─ NamedNodeMap
                └─ Attr (idioma="cat")
            └─ Text (La influència...)
```

El que representa cada una d'aquestes classes és força evident, els noms són prou significatius. Pel que fa als mètodes i atributs que té cada una, no hi entrarem en

profunditat. Bàsicament es tracta de mètodes per a poder navegar i accedir a la informació de cada node. És recomanable disposar d'una bona documentació que permeti d'anar consultant quins mètodes i atributs té cada classe que s'hagi d'utilitzar. L'avantatge que presenta aquesta representació és que en ser una jerarquia d'objectes estàndard, un cop es coneixen aquestes classes, ja se sap utilitzar qualsevol analitzador XML en qualsevol llenguatge de programació. 

Un exercici que pot servir per a entendre la jerarquia DOM és, partint d'un document XML, construir-ne l'arbre XML (s'ha vist la manera de fer-ho en l'apartat "L'arbre XML") i, per a cada node, indicar quina classe DOM el representaria. Després, podeu sofisticar l'exercici plantejant-vos algorismes per a accedir, mitjançant els mètodes i atributs de la jerarquia DOM, a determinada informació emmagatzemada dins de l'arbre.

Per a fer aquest tipus d'exercicis va bé disposar del diagrama de classes en UML de DOM. Aquest diagrama és molt útil per a fer-se una primera imatge mental de tota la jerarquia:



WEB
Unified modeling language (UML) és un llenguatge de modelatge de programari proposat com a estàndard ISO per l'OMG (Object Management Group). Per a més informació consulteu: <http://www.omg.org>

Per acabar, parlarem breument de la classe `Node`, que és la classe genèrica que representa cada un dels nodes d'un arbre XML (fins i tot els atributs) i que, per tant, conté la majoria de mètodes que permeten de navegar per l'estructura.

Mètodes i atributs més importants de `Node`:

- `getNodeType ()`: valor numèric que representa el tipus de node.
- `getNodeName`: nom de l'element. El seu valor depèn del tipus de node.
- `nodeValue`: valor de l'element. Té mètodes `get` i `set`.
- Mètodes d'accés a altres nodes: `getFirstChild()`, `getLastChild()`, `getNextSibling()`, `getChildNodes()`, etc.
- `getOwnerDocument ()`: retorna el node arrel del document.
- `getAttributes ()`: llista d'atributs associats a una etiqueta.
- `hasChildNodes ()`: retorna un booleà que pren per valor cert si l'element té nodes fills i fals si no en té.

getNodeName . Exemple

- Si el node és una etiqueta, el valor d'aquesta propietat serà el nom de l'etiqueta.
- Si el node és un atribut, el valor d'aquesta propietat serà el nom de l'atribut.

3.2.2. Tractament d'errors

La detecció i el tractament d'errors durant el procés d'anàlisi del DOM pot ser força diferent d'un analitzador a un altre. En aquest sentit, l'especificació de nivell 3 de DOM hauria d'aportar-hi molta llum ja que incideix sobre tot allò que té a veure amb el procés de generació del DOM. Malauradament, ara per ara, els analitzadors donen un suport poc fiable al nivell 3 de DOM.

Malgrat això, hi ha un denominador comú en la majoria d'anàlitzadors: per a construir el DOM molts analitzadors utilitzen SAX, és a dir, la majoria d'anàlitzadors fan una anàlisi del document mitjançant SAX i construeixen un arbre DOM a mesura que es van produint els esdeveniments de SAX.

WEB

A l'aula de l'assignatura hi trobareu el fitxer de codi **DOMParser2.java**. Aquest fitxer conté un exemple de crida a un analitzador DOM que capture excepcions SAX. Aquest exemple el podeu utilitzar amb l'anàlitzador Xerces.

Per a tots aquests analitzadors, el tractament d'errors es pot fer capturant les excepcions que genera SAX quan troba errors. !

3.2.3. Disseny

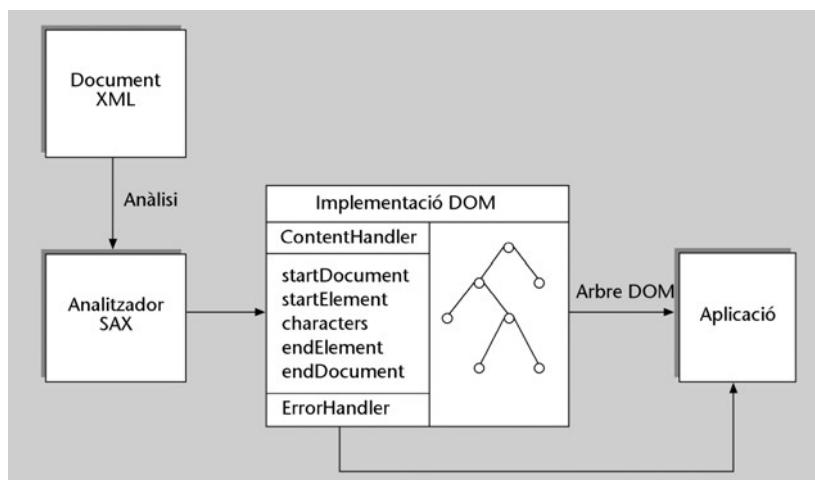
Un analitzador sintàctic XML basat en DOM consta bàsicament de dues parts:

- L'anàlitzador pròpiament dit, que s'encarrega de llegir el document i crear l'arbre DOM.
- Un conjunt de classes de suport que implementen la jerarquia DOM, (package `org.w3c.dom`) i permeten de navegar i accedir a la informació emmagatzemada en memòria.

La manera de programar aquestes classes de suport queda fora de l'abast d'aquest mòdul. El que farem és utilitzar la implementació que en fa el programa Xerces: es creará, per tant, un analitzador però l'arbre DOM serà un arbre DOM de Xerces.

A l'hora de dissenyar un analitzador sintàctic XML basat en DOM podem partir de dos supòsits molt diferents:

- Ja es disposa d'un analitzador XML basat en SAX. En aquest cas, construir una estructura DOM consisteix a redefinir els mètodes de la interfície `ContentHandler` per tal d'anar generant, a mesura que es van produint els esdeveniments, l'arbre DOM:



- No es disposa de cap analitzador, es parteix de zero. En aquest cas s'hi podran aplicar les tècniques de construcció de compiladors que s'han explicat. Caldrà desenvolupar un analitzador lèxic i un de sintàctic, i ho farem, altre cop, utilitzant les eines CASE de construcció d'analitzador lèxics i sintàctics.

El nostre disseny pren com a punt de partida el segon supòsit; en aquest cas, l'analitzador sintàctic que hem d'obtenir és una classe Java, anomenada `DOMParser`, que ha de disposar bàsicament d'un mètode `parse()` i d'un mètode `getDocument()`:

```
class DOMParser {
    private org.w3c.dom.Document doc;

    public parse(String url) {
        // haurà de cridar el mètode d'anàlisi del nostre analitzador.
    }
    public org.w3c.dom.Document getDocument() {
        return doc;
    }
    ...
}
```

 Partirem dels mateixos supòsits i la mateixa gramàtica que s'han utilitzat per a crear l'analitzador sintàctic SAX de l'apartat 3.1.3.

Fet això podem atacar ja el pas 3, que consisteix bàsicament a fer el tractament d'errors i crear l'estructura DOM.

Pel que fa al tractament d'errors, l'especificació DOM no incideix gaire en la manera en què s'ha de fer: per tant, som lliures d'implementar el nostre propi gestor d'errors. És recomanable que els errors es notifiquin mitjançant excepcions Java i que informin al màxim de quin ha estat el problema i on s'ha produït. Una altra opció és generar els errors mitjançant la interfície `ErrorHandler` de SAX, tal com ho hem fet en la implementació de l'analitzador SAX.

No ens entretindrem a crear cap gestor d'errors, és un bon exercici que podeu fer vosaltres mateixos. En lloc d'això passarem directament a la generació de l'arbre DOM a partir de la gramàtica:

<pre>document → elem {doc.appendChild(\$1);} elem → emptyTag {\$\$= \$1;} startTag contingut endTag {\$\$= \$1;} emptyTag → MES_PETIT NOM p_aux atributs BARRA MES_GRAN {\$\$= \$3; } startTag → MES_PETIT NOM p_aux atributs MES_GRAN {\$\$= \$3; } p_aux → {\$\$= doc.createElement("\$0");} endTag → MES_PETIT BARRA NOM MES_GRAN atributs → NOM IGUAL CADENA {\$\$= \$0;} atributs { \$0.setAttribute(\$1, \$3); \$\$= \$0; } ε contingut → elem_aux contingut pcdata_aux contingut ε</pre>	<p>Notació dels atributs</p> <p>Hem especificat els atributs dels símbols de la gramàtica amb la notació de YACC. \$\$ fa referència a l'atribut del símbol de la part esquerra d'una producció. \$1, \$2, etc. fan referència als símbols de la part dreta de la producció (\$1 correspon al primer, \$2 al segon i així successivament). \$0, \$-1, \$-2, etc. fan referència als atributs heretats (\$0 correspon a l'atribut del símbol que s'ha calculat just abans d'analitzar-se el símbol de la part esquerra de la producció actual, \$-1 és l'anterior a aquest i així successivament).</p>
--	--

```

elem_aux      → elem  {$0.appendChild($1); $$= $0; }
pcdata_aux   → PCDATA
{
    $0.appendChild(doc.createTextNode($1));
    $$= $0;
}

```

En alguns llocs s'han utilitzat atributs heretats. La particular estructura de construcció dels arbres DOM ha obligat a fer-ho. Tot seguit veurem JDOM, una API especial Java que permet de construir estructures DOM d'una manera més còmoda. Pot ser un bon exercici reescriure la gramàtica constraint l'arbre mitjançant JDOM. Igualment, també pot ser un bon exercici provar de completar aquest codi i introduir-lo en una eina de creació d'anàltzadors sintàctic. També podeu provar d'ampliar la gramàtica amb altres produccions i fins i tot d'intentar afegir-hi validació: de tota manera, això complica força el problema i hi afegeix la necessitat d'utilitzar una taula de símbols.

3.2.4. Usos de DOM

DOM requereix que es carregui el document sencer en memòria i s'emmagatzemi en una estructura d'arbre. Per aquesta raó, l'ús de DOM requereix una quantitat de memòria proporcional a la mida i a la complexitat del document XML.

També cal tenir en compte que algunes de les operacions sobre estructures d'arbre són cares i comporten una despresa temporal important.

En general l'ús de DOM en documents grans no és gaire recomanable: documents XML com ara llibres o manuals són poc tractables mitjançant DOM.

En Java hi ha una alternativa a DOM que té el propòsit d'ofrir una solució d'alt rendiment tant en l'anàlisi de documents XML com en la posterior manipulació de l'estructura DOM en memòria, l'API JDOM.

3.2.5. JDOM

JDOM és una iniciativa de programari lliure, creada per Brett McLaughlin i Jason Hunter amb el suport de James Duncan Davidson, i en constant evolució.

La idea d'un paquet tan dinàmic com aquest és poder-hi integrar tan ràpidament com sigui possible tot allò que pot fer millorar el rendiment en l'anàlisi i el processament de l'estructura DOM. 

Podeu trobar tota la informació i el programari referent a JDOM a:
<http://www.jdom.org>

WEB

L'API DOM està especificada en CORBA IDF i, per tant, és una especificació independent del llenguatge, mentre que JDOM és una API exclusivament Java i, per tant, intenta explotar al màxim les prestacions que aquest llenguatge li ofereix.

Exemple

JDOM no retorna objectes com NodeList o Attributes, que estan pensats per a oferir certa independència del llenguatge de programació, sinó que retorna directament objectes List i Map, que són específics de Java.

Els mètodes de la jerarquia de classes JDOM són molt més intuïtius i fàcils d'utilitzar en Java que no pas els de DOM i, a més, JDOM permet de connectar-se amb altres analitzadors i fer traduccions d'arbres JDOM a arbres DOM.

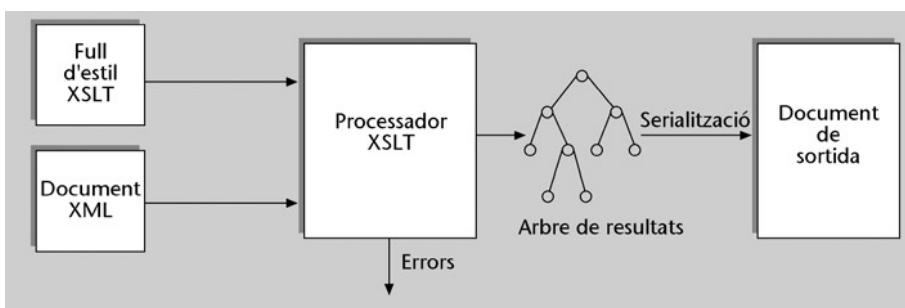
Si es programa en Java, JDOM ofereix més rapidesa i un codi més clar, però es perd l'oportunitat de conèixer a fons una jerarquia d'objectes disponible en tots els llenguatges de programació com és DOM, cosa especialment important si no solament es programa en Java, sinó que eventualment també s'utilitzen altres llenguatges de programació.

Si el rendiment de l'aplicació és un factor summament crític, encara queda una altra alternativa: es pot fer una implementació a mida d'un analitzador per a les tipologies de documents XML amb què s'hagi de treballar. Les tècniques ja les coneixeu, les heu estudiat tant a *Compiladors I* com a *Compiladors II*. Un analitzador específic per a una tipologia de documents XML pot ser més ràpid que no pas un de genèric com ara SAX, DOM i JDOM.

4. Processadors XSLT

Ja hem vist que l'especificació XSL *transformations* permet de definir un conjunt de regles que indiquen com s'ha de transformar cada un dels elements d'un document XML.

Les regles XSLT són processades per programes anomenats *processadors XSLT* que tenen com a entrades un document XML i un document XSLT. Els processadors XSLT apliquen les instruccions definides en el document XSLT sobre un document XML i generen com a sortida un arbre de resultats que es pot serialitzar per a convertir-lo en un document de sortida:



Un processador XSLT és un **traductor***, amb la particularitat que se li pot definir, mitjançant un llenguatge de transformació (XSLT), els criteris de la traducció. I això es pot fer perquè l'entrada (el document XML) no és un llenguatge de programació sinó un llenguatge de marques.

* Programa que llegeix un codi escrit en un llenguatge font i el converteix en un codi escrit en un llenguatge objecte.
Aquest concepte es va estudiar a *Compiladors I*.

Recordeu que un llenguatge de marques és una tecnologia passiva, l'objectiu de la qual és representar l'estructura de la informació amb independència de la informació.

Transformar llenguatges de marques consisteix, bàsicament, a filtrar la informació, reestructurar-la i formatar-la. D'altra banda, és assumible crear un llenguatge per a aquest propòsit (XSLT) i un enginy (el processador XSLT) capaç de processar-lo. En canvi, seria molt més complex plantejar-s'ho per a llenguatges de programació.

Exemple

Es disposa del document XML següent:

```
<?xml version="1.0"?>
<mail>
  <from>pep@uoc.edu</from>
  <to>maria@uoc.edu</to>
  <subject>Hola</subject>
  <message>Bon dia i bona hora.</message>
</mail>
```

Es vol transformar aquest document en la pàgina HTML següent:

```
<html>
  <head>
    <title>Mail</title>
```

```

</head>
<body>
  <p><b>De:</b> pep@uoc.edu</p>
  <p><b>A:</b> maria@uoc.edu</p>
  <p><b>Tema:</b> Hola</p>
  <p><b>Missatge:</b></p>
    <p> Bon dia i bona hora.</p>
</body>
</html>

```

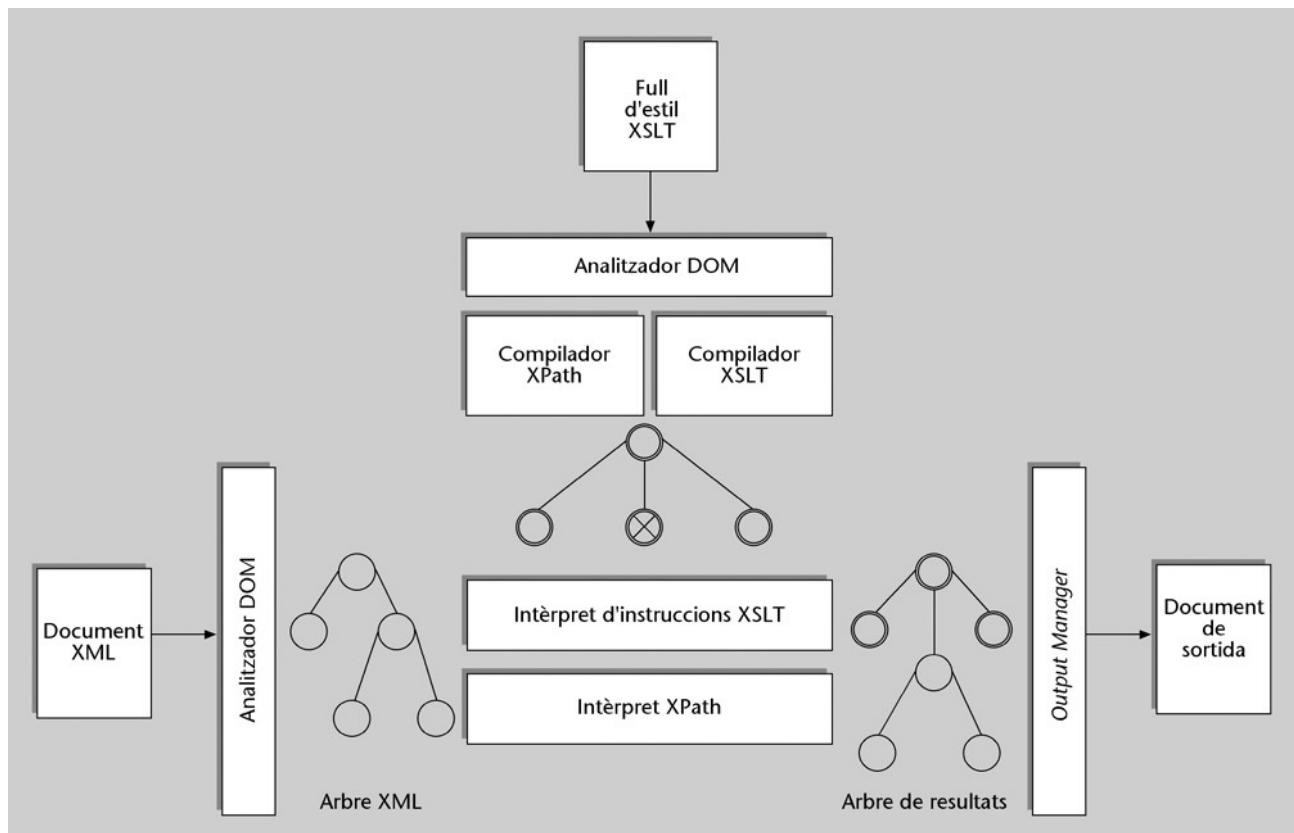
La manera tradicional de fer la transformació plantejada en l'exemple consistiria a construir un programa que llegís el document original (el document XML) i generés la sortida esperada (el document HTML). Amb XSLT la solució se simplifica: només cal definir un full d'estil XSLT i disposar d'un processador XSLT per a aplicar-la sobre l'XML d'entrada.

XSLT elimina la necessitat d'escriure programari a mida per a transformar dades.

Pot no semblar gran cosa, però penseu en la quantitat de vegades que cal transformar dades en les aplicacions informàtiques: fitxers de configuració, entrades per formulari, persistència de dades, pas de missatges entre components, etc.

4.1. Disseny

L'arquitectura interna d'un processador XSLT és una combinació de molts components que ja coneixem:



El document XML i el full d'estil XSLT es carreguen en memòria utilitzant un analitzador sintàctic XML basat en DOM. En el cas del full d'estil, els **nodes encerclats** corresponen a nodes que es copiaran directament a l'arbre de resultats i els **nodes encerclats i marcats amb una creu** corresponen a nodes que contenen instruccions per a aplicar sobre un o diversos nodes de la font XML.

El funcionament és força simple, es va recorrent cada node de l'arbre DOM del full d'estil:

- Quan es troba un **node encerclat**, es copia directament en l'arbre de resultats.
- Quan es troba un **node encerclat i marcat amb una creu** es processa de la manera següent:
 - S'activa l'intèrpret d'XPath, que s'encarrega de fer la cerca sol·licitada sobre l'arbre de la font XML.
 - S'activa l'intèrpret d'instruccions XSLT, que aplica les transformacions sobre els elements que li proporciona l'intèrpret XPath en el pas anterior i copia els nodes resultants en l'arbre de resultats.
- L'*output manager* s'encarrega de serialitzar l'arbre de resultats en un fitxer.

El principal problema dels processadors XSLT és el seu rendiment. 

Mentre que tant l'anàlisi lèxica com la sintàctica estan ben resolts, atès que les tècniques de construcció dels analitzadors XML són les mateixes que les dels compiladors que han estat estudiats i millorats durant anys, el procés de transformació no està optimitzat. Aplicant tècniques avançades d'optimització es podria arribar fàcilment a reduir el cost de processament en un 50%. Actualment es fan importants estudis en aquesta direcció.

4.2. Exemple d'ús d'un processador XSLT

Per a desenvolupar aquest exemple s'utilitzarà el processador XSLT Xalan d'Apache, un producte de programari lliure totalment gratuït i dels més utilitzats pels desenvolupadors de solucions XSLT.

Podeu trobar el processador XSLT Xalan i tota la documentació associada en l'adreça: <http://www.apache.org>.

WEB

Tot i que en aquesta assignatura s'utilitzi el processador XSLT en Java, cal tenir present que hi ha processadors XSLT disponibles en una gran varietat de llenguatges de programació diferents.

En aquest cas, el processador XSLT rep com a entrada el document XML i el document XSLT, i genera com a sortida un document HTML.

Aquest exemple es reprèn en l'annex 3 d'aquesta mateixa assignatura.



- Crida des de línia de comandes:

```
java org.apache.xalan.xslt.Process -in mail.xml -xsl
mail.xslt -out mail.html
```

- Entrada XML (`mail.xml`)

```
<?xml version="1.0"?>
<mail>
  <from>pep@uoc.edu</from>
  <to>maria@uoc.edu</to>
  <subject>Hola</subject>
  <message>Bon dia i bona hora.</message>
</mail>
```

- Entrada XSLT (`mail.xslt`)

```
<xsl:stylesheet version="1.0"
  xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Mail</title>
      </head>
      <xsl:apply-templates select="mail"/>
    </html>
  </xsl:template>
  <xsl:template match="mail">
    <body>
      <xsl:apply-templates select="*"/>
    </body>
  </xsl:template>

  <xsl:template match="from">
    <p><b>De:</b> <xsl:value-of select="."/></p>
  </xsl:template>
  <xsl:template match="to">
    <p><b>A:</b> <xsl:value-of select="."/></p>
  </xsl:template>
  <xsl:template match="subject">
    <p><b>Tema:</b> <xsl:value-of select="."/></p>
  </xsl:template>
  <xsl:template match="message">
    <p><b>Missatge:</b></p>
    <p><xsl:value-of select="."/></p>
  </xsl:template>
</xsl:stylesheet>
```

- Resultat HTML (mail.html)

```
<html>
  <head>
    <title>Mail</title>
  </head>
  <body>
    <p><b>De:</b> pep@uoc.edu</p>
    <p><b>A:</b> maria@uoc.edu</p>
    <p><b>Tema:</b> Hola</p>
    <p><b>Missatge:</b></p>
    <p> Bon dia i bona hora.</p>
  </body>
</html>
```

5. L'API JAXP

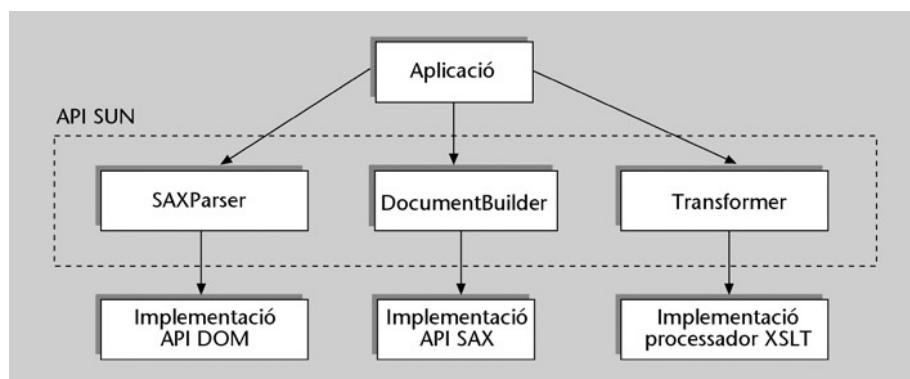
L'any 2000 el nombre d'analitzadors sintàctics XML i de processadors XSLT començava a ser elevat, molts havien nascut com a iniciatives d'investigació dins d'empreses, universitats, etc. Sovint era difícil saber quines funcionalitats i prestacions oferia cada un (alguns no permetien espais de noms, d'altres no suportaven alguns elements d'XSLT, d'altres tenien un pèssim rendiment, d'altres col·lapsaven la memòria, etc.). Decidir-se per una eina o una altra era francament difícil.

Actualment, es considera que tant els analitzadors XML com els processadors XSLT són estables (funcionen bé i suporten tots els estàndards XML i XSLT). De tota manera, aquestes eines han de continuar evolucionant pel que fa a l'optimització del seu rendiment.

Sun Microsystems, tenint en compte que aquestes eines estaven encara en evolució, en lloc de crear les seves pròpies implementacions i entrar en competència amb les que ja hi havia, va crear JAXP (*Java API for XML processing*): una API Java que oferia una única manera de connectar les aplicacions Java amb qualsevol analitzador sintàctic XML i processador XSLT. Sun ofereix als programadors un conjunt d'interfícies, mentre que cada fabricant ofereix les classes que implementen aquestes interfícies i que encapsen el seus productes.

Un símil

JAXP és aproximadament per a les eines de tractament XML el que és JDBC per a les bases de dades.



La part remarcada correspon a l'API de Sun, mentre que les implementacions (la part inferior), les posa cada fabricant.

JAXP ofereix una capa de connectivitat amb qualsevol analitzador XML i processador XSLT, i independitza les aplicacions respecte de les eines d'anàlisi i tractament XML que s'utilitzin.

WEB

Podeu trobar l'última versió de JAXP i la documentació associada a: <http://java.sun.com/xml>.

JAXP permet de canviar l'analitzador XML o el processador XSLT que usa una aplicació per un altre sense haver de recompilar el codi. Si mireu exemples d'ús de JAXP, veureu que el seu codi està completament aïllat de qualsevol detall específic del fabricant de l'analitzador i/o processador.

El conjunt d'enginys vistos al llarg del mòdul s'haurien d'integrar dins les aplicacions Java empresarials mitjançant JAXP. 

JAXP suporta DOM nivells 1 i 2, SAX2 i XSLT 1.0.

A l'aula de l'assignatura hi trobareu exemples del codi Java necessari per a cridar, mitjançant JAXP, un analitzador SAX, un analitzador DOM i un processador XSLT.

WEB

6. Usos i aplicacions de la tecnologia XML

És difícil fer una llista de tots els possibles usos i aplicacions d'XML i els seus derivats. XML és fruit de la investigació i l'evolució natural dels aspectes de la informàtica que incideixen en l'intercanvi de la informació, el seu tractament i la seva estructuració. D'altra banda, incideix en aquest camp i és en tot allò que fa referència a aquesta part de la informàtica on pot tenir els seus usos i aplicacions. Que esdevingui un estàndard per a l'intercanvi de dades, la representació de documents electrònics, la construcció de fitxers de configuració d'aplicacions, els cercadors web, etc., depèndrà de l'evolució de la informàtica, dels nous descobriments que es produixin, de l'acceptació a l'empresa i entre els professionals, de l'ús coherent o no que se'n faci i d'infinitat de factors més.

Per a saber si les tecnologies XML són una bona solució per a resoldre un problema determinat, no hi ha llistes ni fòrmules màgiques: cal analitzar les aplicacions i el seu propòsit, i prendre una decisió. Com sempre, encertar o no depèndrà del coneixement que es tingui de les diferents alternatives.

Malgrat això, no volem acabar sense mostrar exemples actuals d'ús de les tecnologies XML.

6.1. Exemple 1. Aplicacions web

El món de les aplicacions web és un conglomerat, de vegades pintoesc, de tècniques, mètodes, models i tecnologies diferents. En general, amb l'arribada de cada nova tecnologia, com ara CGI, JSP, motors de plantilles, API diverses de programació web, XML i XSLT, etc., han aparegut diferents propostes d'arquitectures de desenvolupament sovint més encaminades a obtenir un benefici comercial que no pas tecnològic.

Malgrat això, hi ha un model universal de disseny per a les aplicacions web, el patró de disseny de programari **model-vista-controlador** (MVC – *model-view-controller*). Aquest és el millor model per a assegurar la separació entre dades, lògica empresarial i presentació.

Hi ha diferents maneres de seguir aquest model en una aplicació web: combinar miniaplicacions de servidor o *servlets* i JSP, utilitzar molts dels *frameworks* de programació per al Web que hi ha avui dia, servidors d'aplicacions, etc., o mitjançant tecnologies XML.

Bibliografia suggerida

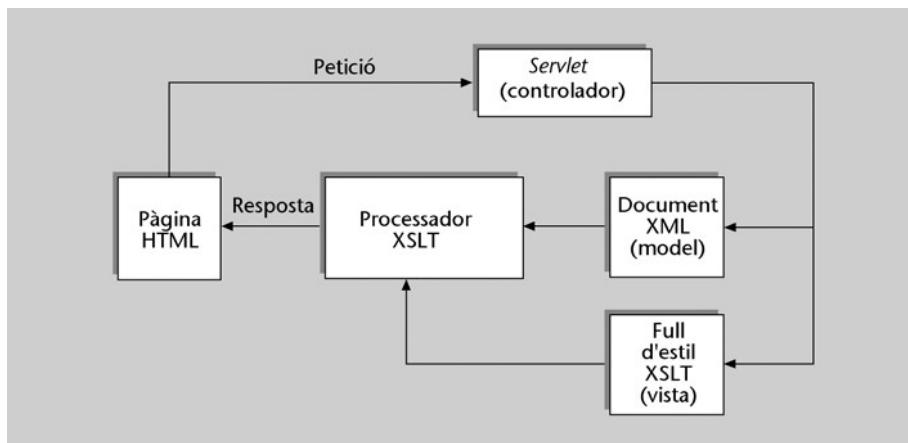
Podeu trobar explicacions detallades d'aquests i d'altres exemples d'aplicacions de les tecnologies XML i XSLT en l'obra següent referenciada en la bibliografia:

Eric M. Burke. *Java and XSLT.*

MVC

La majoria de tècniques efectives per a desenvolupar aplicacions web es basen en el model de disseny **MVC**. Això fa pensar en **MVC** com en una tècnica universal per al desenvolupament de la capa web de les aplicacions.

L'arquitectura d'una aplicació web basada en XML que satisfà el patró de disseny MVC és la següent:



El document XML representa el model, la miniaplicació de servidor representa el controlador i el full d'estil XSLT, la vista. És important fer notar que el full d'estil també pot contenir lògica de programació, ja que podria incloure *scripts* en javaScript, codi JSP, ActiveX etc. En tot cas, és responsabilitat de l'equip de desenvolupament desdibuixar més o menys la línia divisòria entre la vista i el controlador.

6.2. Exemple 2. Rendiment

Una de les crítiques més importants que es fa a la tecnologia XML fa referència al seu rendiment: es diu que el temps de processar un XML, un XSL i generar una sortida en un altre format és massa elevat.

Tot i que això és cert, no ho és menys que el nombre de documents XSLT creats correctament és força baix: massa sovint, en un document XSLT hi ha expressions XPath molt poc optimitzades que fan augmentar el temps de processament molt més del que caldría. També és veritat que els fabricants de processadors XSLT han treballat poc en l'optimització del rendiment dels seus productes i que s'espera que en els pròxims anys les eines millorin molt en aquest aspecte.

Amb tot, hi ha tot un ventall de tècniques que permeten de treballar amb aquestes tecnologies i aconseguir rendiments més que acceptables. En destacerem dues:

- Desactivar la validació: la majoria d'analitzadors sintàctics XML permeten de desactivar la validació. Amb això es deixa de comprovar si els documents XML s'ajusten als seus DTD, però s'obtenen guanys importants en rendiment. A més, tot i que la validació va molt bé per al desenvolupament i les proves de l'aplicació, un cop aquesta ja funciona correctament no és gaire necessària.
- Emmagatzemar fulls d'estil XSLT en cau. Es tracta de tenir el document XSLT sempre carregat en memòria en forma d'arbre XML, cosa que estalvia el temps d'anàlisi sintàctica. Es calcula que amb aquesta tècnica es poden aconseguir guanys de rendiment significatius (aproximadament del 30%) en les velocitats dels processadors XSLT. A vegades, de-

penent de l'aplicació, és possible tenir en memòria tant el document XSLT com el document XML.

6.3. Exemple 3. Internacionalització (i18n)

Una de les tècniques en què s'utilitza la tecnologia XSLT avui dia és per a la internacionalització (i18n); és a dir, per a oferir suport a diferents idiomes.

El terme *internacionalització* fa referència als aspectes que cal tenir en compte per a construir una aplicació en què sigui possible canviar l'idioma de la interfície de comunicació amb l'usuari.

Per convenció, s'utilitza la paraula **i18n** per a fer referència a la internacionalització. El mot té un origen curiós, 18 és el nombre de lletres que hi ha entre les lletres *i* i *n* en la paraula anglesa *internationalisation*.

Per exemple, si es vol construir una pàgina web amb suport a dos idiomes: català i xinès. La tècnica consisteix a crear un full d'estil XSLT que generi la pàgina en un dels idiomes (per exemple, en català) que contingui variables amb tots i cada un dels literals que han d'aparèixer a la pàgina.

Per a crear la mateixa pàgina en xinès, només cal crear un altre full d'estil XSLT que importi el primer (amb l'element `<xsl:import>`). En aquest segon full, es canvia la codificació de la sortida (amb l'element `<xsl:output>`) per UTF-16, que permet de representar caràcters en xinès; es tornen a definir les mateixes variables per als literals, aquest cop escrits en xinès, i es redefineixen tots aquells `<xsl:templates>` que calgui per a adaptar l'aparença de la pàgina al nou idioma.

Tots els elements del full d'estil importat tenen menys prioritat que els del principal; per tant, prevaldran les definicions de variables del full d'estil principal (el del xinès) sobre les de l'importat (el del català). I el mateix passarà amb els `<xsl:templates>` definits i amb qualsevol altre element.

Resum

En aquest mòdul s'han estudiat conceptes relacionats amb els llenguatges de marques. S'ha parlat de les tecnologies XML i s'ha vist la relació que hi ha entre la teoria dels compiladors i les eines de tractament de documents XML.

Probablement, poques vegades us haureu d'enfrontar amb el problema de crear una eina de tractament de documents XML, però el fet de conèixer les tècniques que s'apliquen per a construir-les és bàsic per a poder treure'n el màxim rendiment.

Abans d'acabar, un recordatori sobre els annexos i els exemples que trobareu a l'aula de l'assignatura: els annexos i els exemples són material de referència pensat especialment com a ajuda per a poder portar a la pràctica el que heu après al llarg del mòdul.

Activitats

1. Construïu un analitzador que permeti de validar la sintaxi d'un DTD utilitzant les tècniques de compiladors explicades.
2. Introduïu la gramàtica que s'explica en l'apartat 3.1.3 a l'eina CASE que utilitzeu per a crear analitzadors sintàctics. Completeu el codi, compileu-lo i proveu-ne el funcionament.
3. Donada la gramàtica de l'apartat 4.2.3, afegiu-hi el que calgui per fer el tractament d'errors.
4. Introduïu la gramàtica que s'explica en l'apartat 4.2.3 a l'eina CASE que utilitzeu per a crear analitzadors sintàctics. Completeu el codi, compileu-lo i proveu-ne el funcionament.
5. Escriviu un programa que utilitzi SAX. Redefiniu el tractament d'errors per tal que quan es produeixi un *warning* o un *error* l'anàlisi no continui, sinó que acabi. Quan el programa acaba per culpa d'un d'aquests errors en l'anàlisi, ha de mostrar per pantalla la línia on s'ha produït l'error, l'URI del document XML analitzat i una descripció del problema trobat.
6. Escriviu un programa que utilitzi DOM i que tingui el funcionament següent: ha de permetre d'analitzar documents XML i, una vegada creat l'arbre DOM corresponent, imprimir el nom dels elements en *postordre*.

Exercicis d'autoavaluació

1. Poseu un exemple de llenguatge de marques i expliqueu breument el tipus de marques que utilitza i què fa cadascuna.
2. Expliqueu per a quin tipus d'aplicacions és útil el marcatge de format i per a quines no. Poseu un exemple d'aplicació o programa en què podria ser útil i un altre en què no. Expliqueu també una possible solució per a l'exemple d'aplicació en què el marcatge de format no serveix.
3. Per a cada un dels dos documents XML següents, marqueu els blocs d'informació que els constitueixen, dibuixeuhne l'arbre XML associat i indiqueu, per a cada node de l'arbre, la classe DOM que el representaria:

```
<llibre>
  <titol>Els Compiladors</titol>
  <autor>Pere Token Yacc</autor>
  <editorial>El Compilador</editorial>
  <any>2003</any>
  <capitols>
    <capitol num="1">
      <titol>Conceptes bàsics</titol>
      <contingut>
        <p>Lorem <b>ipsum</b> ...</p>
        <p>Lorem ipsum ...</p>
      </contingut>
    </capitol>
    <capitol num="2">
      <titol>Exemples pràctics</titol>
      <contingut>
        <p>Lorem ipsum ...</p>
        <p>Lorem <b>ipsum</b> ...</p>
      </contingut>
    </capitol>
  </capitols>
</llibre>

<missatge>
  <titol>Reunió a les <ressaltat>16:00 hores</ressaltat></titol>
  <cos>
    bon dia. La <ressaltat>reunió <titol id="25">notes de compiladors</titol> </ressaltat> que hi havia convocada per a les 15:00 hores es posposa fins a les <ressaltat>16:00 hores</ressaltat>.
    <signatura dni="00000000-L"/>
  </cos>
</missatge>
```

4. Expliqueu, en paraules, quins elements pot contenir l'element `llibre` en cada una de les tres definicions següents:

- a) `<!ELEMENT llibre (index, capitol+)>`
- b) `<!ELEMENT llibre ((index, capitol+), (index, capitol+)**)>`
- c) `<!ELEMENT llibre ((index, capitol+) | (index, capitol+)**)>`

5. Busqueu, amb les fonts d'informació que han aparegut al llarg del mòdul, el significat del tipus de dades CDATA. Expliqueu breument què permet de fer i poseu un exemple de document XML que el contingui.

6. Identifiqueu quines de les afirmacions següents són falses i expliqueu breument el perquè:

- a) Els documents XML estan ben formats i són vàlids.
- b) Un document XML que no té cap DTD o esquema associat és invàlid.
- c) Un document XML és més estructurat que un document PDF.
- d) Una etiqueta XML és el mateix que un element XML.

7. Indiqueu els esdeveniments que es produirien i el caràcter que llegiria l'analitzador en el moment de llançar cada esdeveniment si s'analitza el document XML següent amb un analitzador SAX:

```
<?xml version="1.0"?>
<doc>
    <hello>Hola, <b>Bon dia</b> i bona hora.</hello>
    <bye>Adéu-siau</bye>
</doc>
```

8. Expliqueu què fan les expressions següents:

- a) `<xsl:value-of select="current()"/>`
- b) `<xsl:value-of select=". "/>`
- c) `<xsl:apply-templates select="//llibre[@titol=current()]/@referencia"/>`
- d) `<xsl:apply-templates select="//llibre[@titol=../@referencia"]"/>`

9. Dibuixe l'arbre XML del document XML d'entrada i el diagrama d'activació de les regles XSLT per al següent exemple de transformació. Per a cada activació, marqueu el node de l'arbre XML que s'explora:

- Entrada. Document XML

```
<?xml version="1.0"?>
<doc>
    <missatge>Hello World!</missatge>
    <autor>Un estudiant d'XSLT</autor>
</doc>
```

- Entrada. Regles XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" omit-xml-declaration="yes"/>
    <xsl:template match="/doc">
        <html>
            <head>
                <title><xsl:apply-templates select="missatge"/></title>
            </head>
            <body>
                <h1><xsl:apply-templates select="missatge"/></h1>
                <p><xsl:apply-templates select="autor"/></p>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="*>">
        <xsl:value-of select=". "/>
    </xsl:template>
</xsl:stylesheet>
```

- Resultat. Document HTML

```
<html>
    <head>
        <title> Hello World!</title>
    </head>
    <body>
        <h1> Hello World!</h1>
```

Nota

En l'apèndix 3 d'aquest mateix mòdul didàctic en trobareu un exemple.

```
<p> Un estudiant d'XSLT</p>
</body>
</html>
```

10. Expliqueu què fa el document XSLT següent:

```
<xsl:stylesheet version="1.0"
xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

11. Amplieu la gramàtica de l'apartat 3.1.3 perquè accepti la definició de document XML:

<?xml version="1.0"?>. Afegiu-hi les accions necessàries perquè es generi error si el valor de version no és 1.0.

Solucionari

1. Un exemple de llenguatge de marques podria ser el següent:

Aquest és un petit exemple de com podria ser un **** document escrit amb un editor basat en <i>marques de format</i>. Conté paraules en negreta i en cursiva.

En aquest cas es disposa d'una marca d'inici *<nom_marca>* i una de final *</nom_marca>*, en què *nom_marca* indica el tipus de format que cal aplicar. En l'exemple, la marca **b** indica negreta i la marca **i** indica cursiva. El resultat d'imprimir aquest document seria el següent:

Aquest és un petit exemple de com podria ser un **document** escrit amb un editor basat en *marques de format*. Conté paraules en negreta i en cursiva.

2. El marcatge de format és molt eficaç quan es vol dur a terme el procés: crear el document, guardar-lo en una representació concreta i imprimir-lo, i, en general, per a tot allò que té a veure amb l'edició de textos. No és gaire indicat, però, per a aplicacions que vulguin cercar parts dels documents, cercar per continguts, recuperar informació segons criteris específics, partir, unir o transformar documents; en general, per a tot allò que té a veure amb la gestió documental.

Exemple d'aplicació en què és útil el marcatge de format: un processador de textos com ara Microsoft Word.

Exemple d'aplicació en què no és útil el marcatge de format:

Imaginem que escrivim en un editor de textos un document que és una entrevista a una actriu de cinema. Podríem decidir marcar amb negreta les preguntes del periodista i deixar com a text normal les respostes de l'actriu. La representació que obtindriem d'aquest document tindria marques que indicarien quan un text està en negreta, però no indicarien el que és una pregunta i el que és una resposta. Guardem aquest document al disc dur.

Després, fem una cerca per Internet per trobar altres entrevistes d'aquesta actriu i guardar-les en una carpeta junt amb l'entrevista que havíem editat. El cercador ens en dóna de molts tipus, unes amb HTML, d'altres amb PDF, d'altres amb RTF, i no solament això, sinó que cada document utilitza uns criteris de format diferents per a presentar què és una pregunta i què és una resposta.

Finalment, volem programar una aplicació que emmagatzemi tots els parells pregunta-resposta en una base de dades i permeti de fer cerques dels que continguin una mateixa paraula.

Per a una aplicació com aquesta el marcatge de format no seria gens indicat, perquè la major part de documents tenen **diferents representacions de la informació** i aquestes representacions descriuen com s'ha de presentar el contingut però no quins components lògics el formen.

La solució per a l'aplicació que acabem de descriure és utilitzar documents marcats amb un **llenguatge de marques generalitzat**, ja que d'aquesta manera l'aplicació podria conèixer quina és l'estructura lògica de cada document.

Aplicant aquest nou paradigma, el que s'hauria de fer és obrir l'editor de textos i començar a reescriure l'entrevista, precedint cada pregunta i cada resposta d'una marca que les identifiqués com a tals.

3.

Primer document:

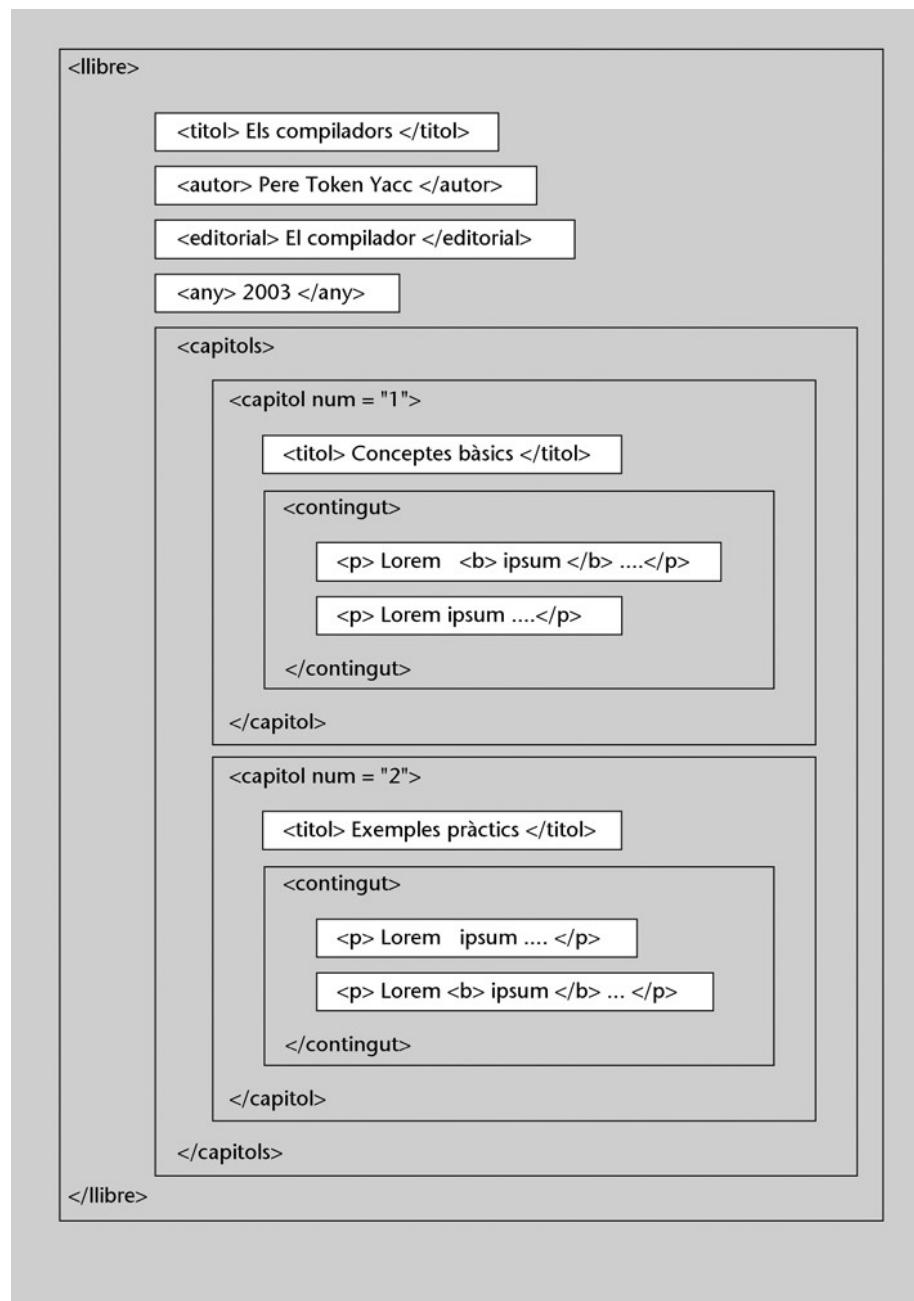
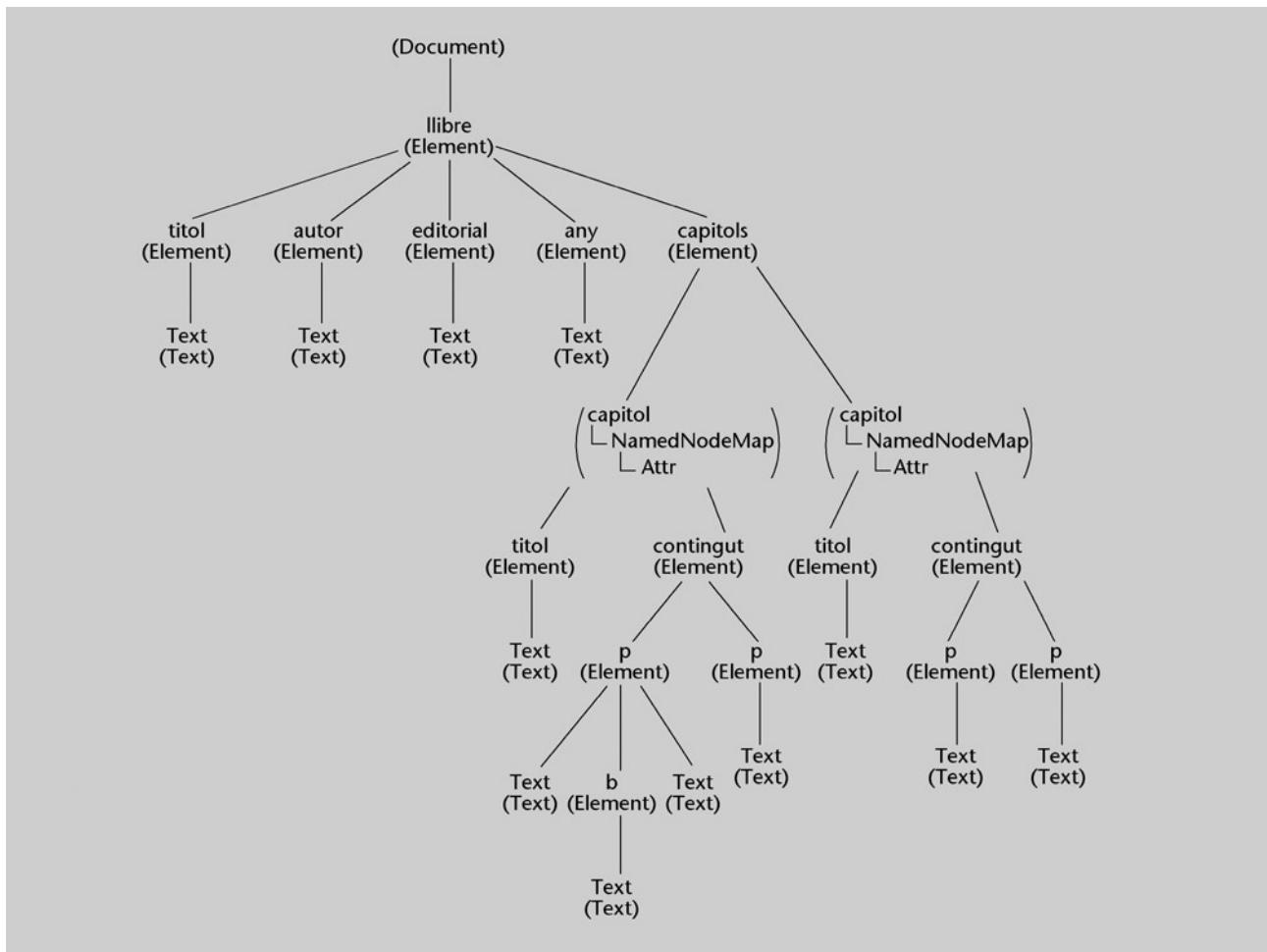


Diagrama de blocs d'informació.



Segon document:

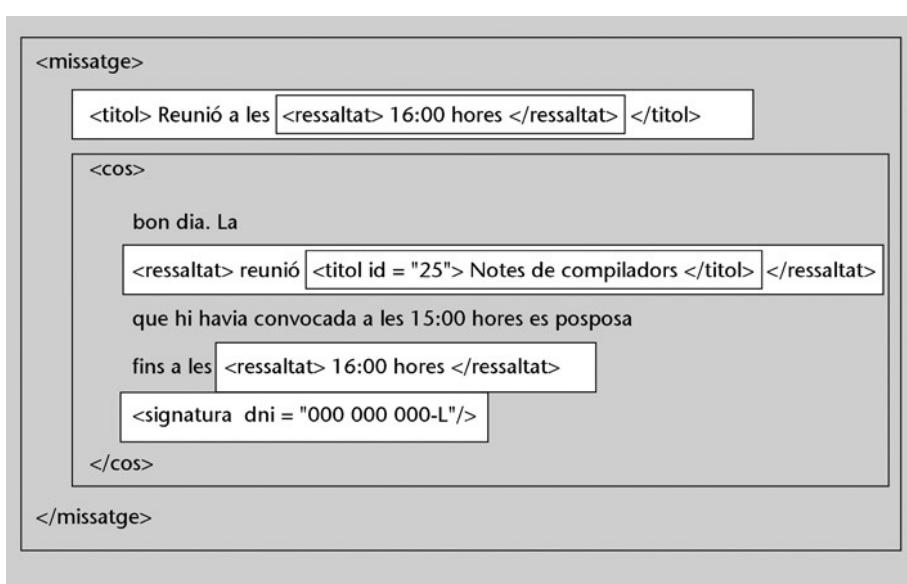
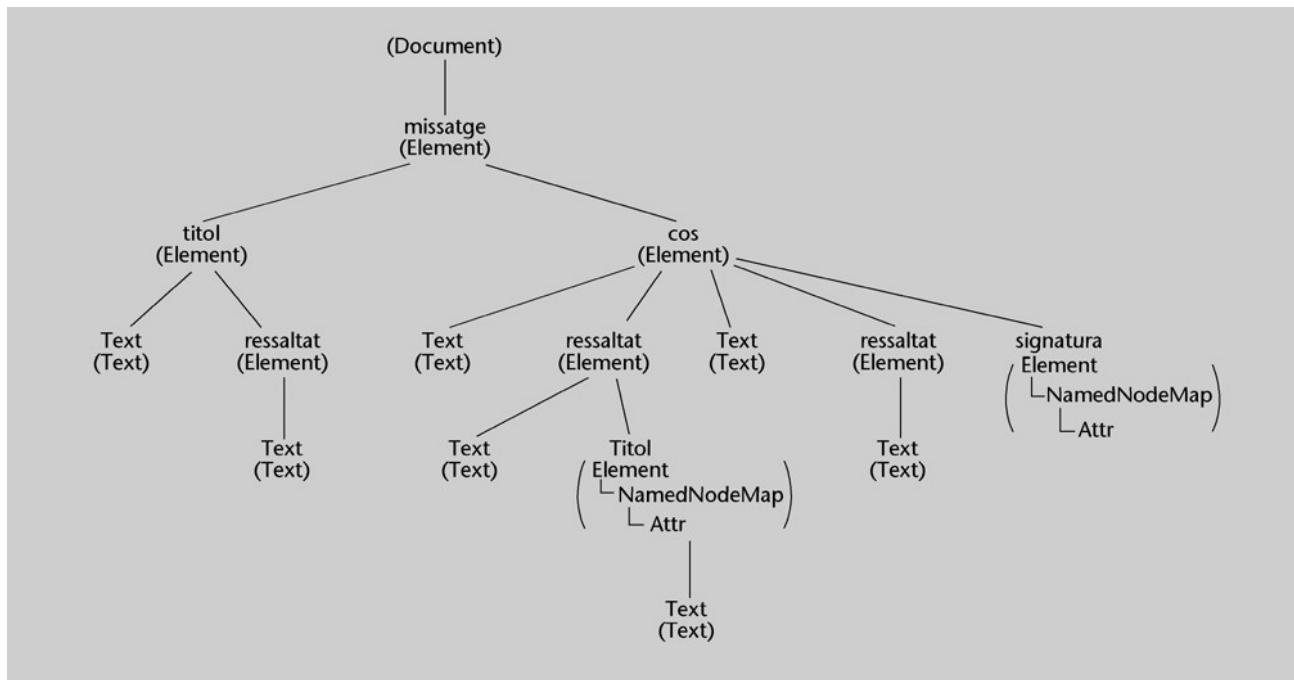


Diagrama de blocs d'informació.



4. L'element `llibre` pot contenir els elements següents:

- a) Un element `index` seguit d'un o més elements `capitol`. Podria servir per a representar un sol llibre.
- b) Una o més aparicions de l'estructura: un element `index` seguit d'un o més elements `capitol`. Podria servir per a representar una enciclopèdia o, en general, llibres de més d'un volum.
- c) El mateix que l'apartat b), amb la particularitat que en aquest cas també es permet que l'element `llibre` estigui buit.

5. Un element amb contingut de tipus CDATA és un element que conté dades que no són analitzades per l'analitzador sintàctic. És a dir, quan l'analitzador reconeix un element CDATA, l'analitzador en consumeix tots els caràcters fins a arribar al final. Això és molt útil si cal introduir dades amb caràcters especials com < i > sense haver-los de codificar.

Una secció CDATA es marca de la manera següent:

`<![CDATA[aquí hi va el contingut]]>`

Exemple:

```
<dades_html>
<![CDATA[ <p> Això és un exemple de <b>CDATA</b> ]]>
</dades_html>
```

6.

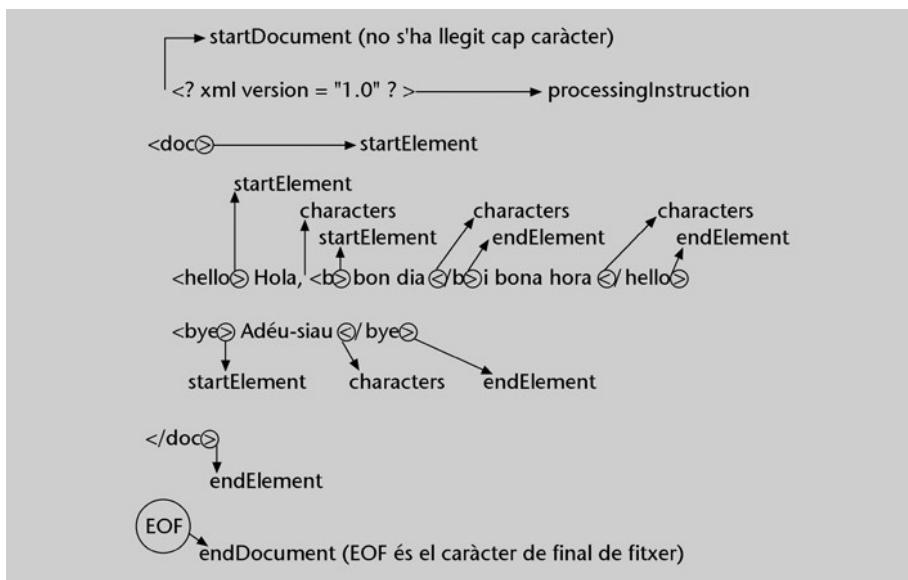
a) Fals. Tots els documents XML estan ben formats i alguns, a més, són vàlids.

b) Fals. No té sentit plantejar-se la validesa o invalidesa d'un document XML que no té cap DTD o esquema associat. De fet, seria més apropiat parlar de documents amb tipus vàlid o documents amb tipus invàlid que no pas simplement de documents vàlids o invàlids. Si un document no és de cap tipus determinat (en altres paraules, no té cap DTD o esquema associat), no es pot dir res sobre la validesa del seu tipus, perquè no en té.

c) Fals. Un document PDF també pot ser estructurat, pot estar compost per pàgines, columnes, taules, etc. La diferència rau en el fet que XML és una representació basada en marcatge generalitzat mentre que PDF, no.

d) Fals. Les etiquetes descriuen els elements però no són elements. És fàcil d'entendre fent un símil amb una carta: l'etiqueta de la carta dóna informació sobre la carta (destinatari, remitent, etc.), però la carta inclou més coses que l'etiqueta; per exemple, un escrit, un fullet informatiu o, fins i tot, una altra carta.

7.



- a) Reescriu a la sortida el contingut (sense les marques) de l'element de l'arbre XML que es tracti.
- b) Fa exactament el mateix que a).
- c) Cerca tots els elements llibre que tenen l'atribut `titol` igual a l'atribut `referencia` del node actual.
- d) Cerca tots els elements llibre que tenen els atributs `titol` i `referencia` iguals. Per exemple:
`<llobre titol="XSLT" referencia="XSLT">...</llobre>`

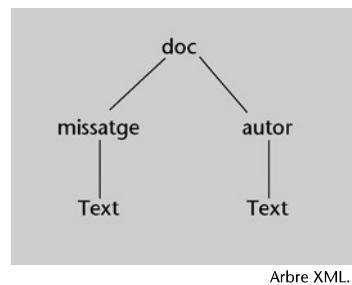
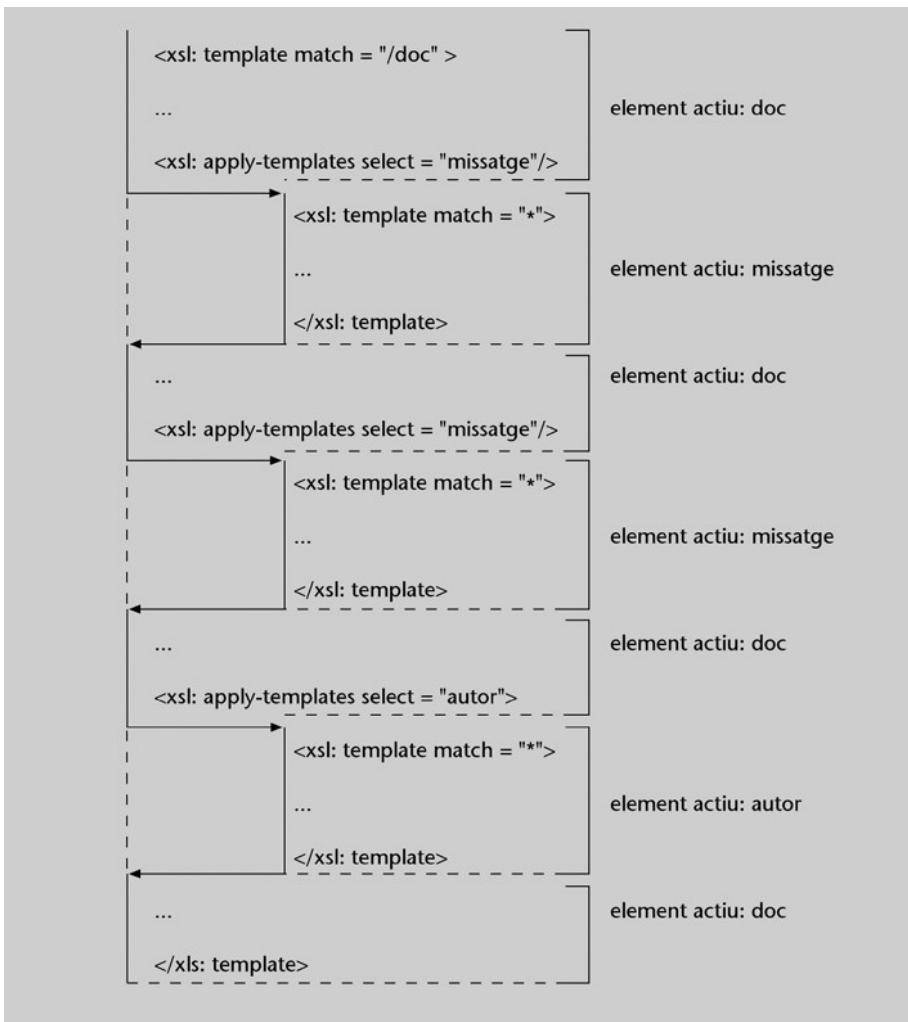


Diagrama d'activacions.

10. El document XSLT s'anomena **transformació identitat** i genera com a sortida el mateix document XML d'entrada.

11.

Nous lexemes:

Token	Expressió regular (LEX)	Descripció
INIDEC	"<?"	Inici d'una declaració
ENDDEC	"?>"	Final d'una declaració

Pel que fa a la gramàtica, només replicarem la part retocada. També caldrà definir una nova producció: **declaracio**.

```

document      →     declaracio    elem
declaracio   →     INIDEC        NOM      atributs  ENDDEC
{
    if($2=="xml" || $2=="XML") {
        Attribute atr= null;
        atr=$3.getAttribute("version");
        if(Atr!=null){
            if(attr.getValue() !=)
                error("versió d'XML no vàlida");
        }
        else error("No s'ha definit la versió
d'XML");
    }
    else error("declaració incorrecta.");
}
|     INIDEC      NOM      error      ENDDEC { yyerrok; }
|     INIDEC      error  ENDDEC { yyerrok; }
atributs →  NOM  IGUAL  CADENA atributs
{
    $4.addAttribute($1, $3);
    $$= $4;
}
|     ε           {$$= new LlistaAtributs();}
```

S'hi han afegit dues produccions de tractament d'error que permeten de fer el control d'errors per a declaracions incomplides, com ara <? xml ?>.

La funció **error()** escriu per pantalla el missatge d'error i la informació de la línia i el caràcter on s'ha produït. **yyerrok;** indica a l'analitzador sintàctic que ja s'ha tractat l'error i que, per tant, es pot continuar l'anàlisi sintàctica.

Annex 1

Sintaxi dels documents DTD

En aquest annex es fa un cop d'ull a la declaració de marcage, la notació que s'utilitza en els documents DTD. És una petita guia de referència que en cap cas no pretén ser exhaustiva.

Podeu trobar informació referent als documents DTD i a la seva sintaxi a <http://www.w3schools.com> i a <http://www.w3.org>.

WEB

Declaració d'elements

<!ELEMENT element-name category>

On category pot prendre com a valors: EMPTY, #PCDATA o ANY.

<!ELEMENT element-name EMPTY>	
Descripció	Permet de declarar elements que no tenen contingut.
Exemple DTD	<!ELEMENT br EMPTY>
Exemple XML	

<!ELEMENT element-name (#PCDATA)>	
Descripció	Permet de declarar elements que tenen com a contingut una cadena de caràcters.
Exemple DTD	<!ELEMENT from (#PCDATA)>
Exemple XML	<from>Pep</from>

<!ELEMENT element-name ANY>	
Descripció	Permet de declarar elements que poden tenir com a contingut qualsevol altre element.
Exemple DTD	<!ELEMENT note ANY>
Exemple XML	<note> ... <from> ... </from> ... </note>

<!ELEMENT element-name (element-content)>.

On element-content especifica les etiquetes que pot contenir l'element element-name. Hi ha un conjunt d'operadors que permeten d'indicar la manera en què pot aparèixer cada element:

Llista d'operadors	
'	Una seqüència. (nom-element1, nom-element2, nom-element3)
	Una or. (nom-element1 nom-element2)

Llista d'operadors	
*	De 0 a N. (nom-element*)
+	D'1 a N. (nom-element+)
?	0 o 1. (nom-element?)

Es poden combinar operadors mitjançant parèntesis:

Exemples

- (to, from, heading, body)
- (to+,from, heading?, body)
- (to+,from, (heading?, body) | (heading, body))

Declaració d'atributs

<!ATTLIST element-name attribute-name attribute-type default-value>	
Exemple DTD	<!ATTLIST payment type CDATA "check">
Exemple XML	<payment type="check" />

Principals valors per a attribute-type	
CDATA	Indica que l'atribut és de tipus <i>string</i> .
(en1 en2 ...)	Indica que l'atribut és de tipus enumeració.
ID	Indica que l'atribut és de tipus identificador (ha de ser únic en tot el document).
IDREF	Indica que l'atribut és de tipus referència a un identificador (d'un altre element).

Principals valors per a default-value	
Value	Indica el valor per defecte que pren l'atribut.
#REQUIRED	Indica que l'atribut és obligatori.
#IMPLIED	Indica que l'atribut no és obligatori.
#FIXED Value	Indica un valor fix per a l'atribut (en l'XML no serà possible canviar-lo).

Annex 2

Expressions XPath

XPath és un llenguatge d'expressions que permet de fer cerques i consultes sobre la informació emmagatzemada dins d'una estructura XML. Cal tenir en compte que XPath **no** és un llenguatge basat en XML.

WEB

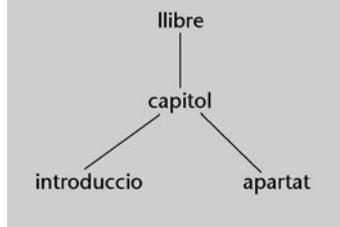
Podeu trobar documentació referent a l'especificació XPath a:<http://www.w3.org>. Podeu trobar manuals i exemples pràctics d'ús d'aquesta especificació a:<http://www.w3schools.com>.

XPath permet d'accendir a un o a diversos nodes de l'arbre XML. Hi ha bàsicament dues maneres d'accendir als nodes:

- Indicant localitzacions absolutes des de l'arrel de l'arbre. La manera d'expressar-les és semblant a la manera d'expressar referències a directoris en una màquina Unix. Seguint aquesta metàfora, l'arrel de l'arbre XML es correspondria amb l'arrel de la màquina i les etiquetes que pengen de l'arrel amb els diferents directoris de la màquina.

Exemple

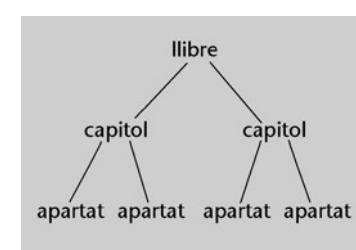
Expressió XPath	Descripció
/llibre/capitol/apartat	Accediria a l'element apartat del capitol del llibre.
/llibre/capitol/*	Accediria a tots els elements que pengen de capitol, que en aquest cas són introduccio i apartat.



Un arbre XML pot tenir molts nodes germans amb el mateix nom, mentre que una estructura de directoris, no (un directori no pot contenir dos directoris que es diguin igual).

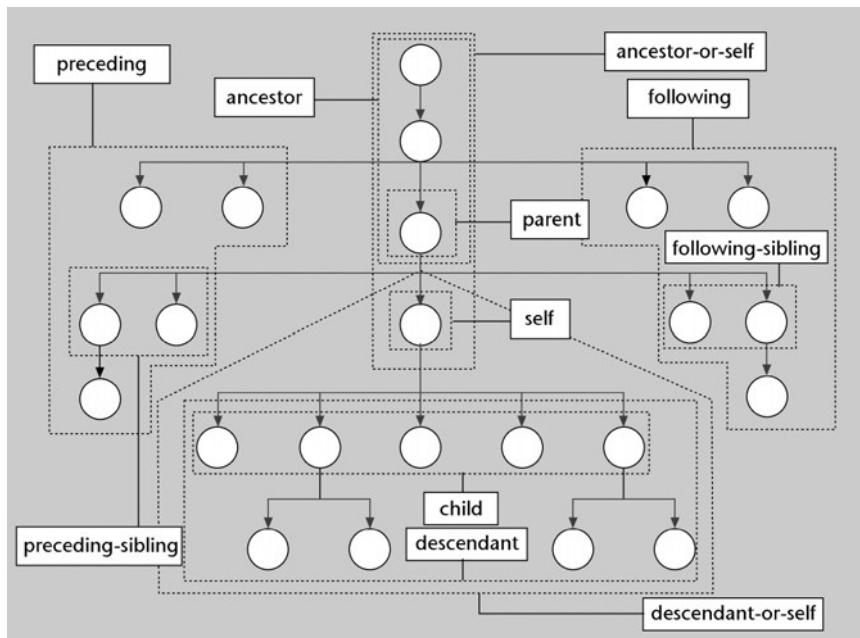
Una expressió XPath en un arbre XML amb nodes germans repetits es comportaria de la manera següent:

Expressió XPath	Descripció
/llibre/capitol/apartat	Accediria a tots els elements apartat de tots els elements capitol del llibre.
/llibre/capitol/*	Accediria a tots els elements que pengen de capitol, que en aquest cas són introduccio i apartat.



- Indicant localitzacions relatives a una posició determinada, a partir de la qual és possible desplaçar-se en diferents direccions segons un conjunt de

modificadors de direcció. L'esquema següent és un mapa d'àrees a què fa referència cada un d'aquests modificadors:



Exemple

Expressió XPath	Descripció
child::*	Selecciona tots els fills del node actual.
child::para	Selecciona tots els fills <code>para</code> del node actual.
attribute::*	Selecciona tots els atributs del node actual.
descendant::para	Selecciona tots els nodes <code>para</code> que són descendents del node actual.
child::capitol/descendant::*	Selecciona tots els descendents dels fills del node actual que són nodes <code>capitol</code> .
preceding-sibling::capitol	Selecciona tots els germans del node actual que apareixen en l'arbre abans que ell i que són nodes <code>capitol</code> .

Hi ha una sintaxi simplificada per a expressar alguns dels modificadors anteriors:

Abreviatura	Modificador equivalent	Exemple
//	descendant	<code>//para</code> és equivalent a <code>descendant::para</code>
.	self	<code>.//para</code> és equivalent a <code>self::*/para</code>
..	parent	<code>...//para</code> és equivalent a <code>parent::*/para</code>
@	attribute	<code>@titol</code> és equivalent a <code>attribute::titol</code>
	child	<code>para</code> és equivalent a <code>child::para</code>

Després d'indicar la localització, una expressió XPath pot contenir una o diverses condicions que permeten d'afinar encara més la cerca. Aquestes condicions s'expressen mitjançant predicats que s'avaluen per a cada node localitzat. Si el resultat de l'avaluació és fals, el node es descarta.

Exemple

Expressió XPath	Descripció	Notació
capitol[1]	Selecciona el primer node <code>capitol</code> fill del node actual.	
capitol[position()=1]	Fa el mateix que l'expressió <code>capitol[1]</code> .	
capitol[last()]	Selecciona l'últim node <code>capitol</code> fill del node actual.	
capitol[@titol="El carrer"]	Selecciona el node <code>capitol</code> que té com a valor de l'atribut <code>titol</code> "El carrer".	Els predicats són expressions que van entre els signes [i]. En l'exemple s'ha marcat en negreta la part de l'expressió XPath que correspon als predicats.

En els predicats es poden utilitzar alguns operadors bàsics i determinades funcions. Tot seguit trobareu una taula que resumeix aquests operadors i una taula que resumeix les funcions més importants.

Operadors

Tipus	Operadors	Descripció
Numèrics	+ - * div mod	Suma Resta Multiplicació Divisió Mòdul
D'igualtat	= != > < ≥ ≤	Igual Diferent Més gran Més petit Més gran o igual Més petit o igual
Booleans	or and	O lògica I lògica

Funcions

Definició	Descripció
<code>count(node-set) : number</code>	Compta el nombre de nodes de <code>node-set</code> .
<code>local-name(node) : string</code>	Retorna el nom de l'element.
<code>name(node) :string</code>	Retorna el nom complet de l'element amb el prefix de l'espai de noms inclòs.
<code>position() : number</code>	Retorna la posició de l'element.
<code>concat(val1, val2, ...): string</code>	Concatenació de <i>strings</i> .
<code>contains(val, substr): string</code>	Retorna cert si <code>val</code> conté el <code>substring</code> <code>substr</code> .
<code>normalize-space(string) : string</code>	Substitueix els caràcters "espai en blanc" consecutius per un de sol.

Definició	Descripció
<code>starts-with(string, substr): string</code>	Retorna cert si <code>string</code> comença per <code>substr</code> .
<code>string-length(string): number</code>	Retorna la mida de l' <code>string</code> .
<code>substring(string, start, length): string</code>	Retorna el substring de <code>string</code> que comença en el caràcter <code>start</code> i té longitud <code>length</code> .
<code>substring-after(string, substr): string</code>	Retorna el substring de <code>string</code> que hi ha després de la primera ocurrència de <code>substr</code> .
<code>substring-before(string, substr): string</code>	Retorna el substring de <code>string</code> que hi ha abans de la primera ocurrència de <code>substr</code> .
<code>translate(value, string1, string2): string</code>	En la cadena <code>value</code> , tots els caràcters de <code>string1</code> se substitueixen pels de la mateixa posició de la cadena <code>string2</code> . Exemple: <code>translate("hola món","hm","HM")</code> retorna: "Hola Món"

Annex 3

El llenguatge de transformació XSLT

Aquest annex és una petita guia de referència XSLT. El seu objectiu és proporcionar les nocions bàsiques i els criteris més importants que cal tenir presents a l'hora d'enfrontar-se amb XSLT.

Partirem d'un exemple de transformació que permetrà d'il·lustrar com s'activen les regles XSLT; després, es descriuràn els elements més importants de l'especificació, i finalment es presentaran alguns consells importants per a desenvolupar bé fulls d'estil XSLT.

WEB

Podeu trobar tota la informació referent a l'estàndard XSLT en l'adreça següent:
<http://www.w3.org>.
 Podeu trobar manuals i exemples pràctics de l'estàndard en l'adreça següent:
<http://www.w3schools.com>.

Activació de templates

Exemple de referència

El processador XSLT rep com a entrada el document XML i el document XSLT, i genera com a sortida un document HTML.

- Crida des de línia de comandes

```
java org.apache.xalan.xslt.Process -in mail.xml -xsl mail.xslt
-out mail.html
```

- Entrada XML (`mail.xml`)

```
<?xml version="1.0"?>
<mail>
  <from>pep@uoc.edu</from>
  <to>maria@uoc.edu</to>
  <subject>Hola</subject>
  <message>Bon dia i bona hora.</message>
</mail>
```

- Entrada XSLT (`mail.xslt`)

```
<xsl:stylesheet version="1.0"
  xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Mail</title>
```

```

</head>
<xsl:apply-templates select="mail"/>
</html>
</xsl:template>
<xsl:template match="mail">
<body>
<xsl:apply-templates select="*"/>
</body>
</xsl:template>

<xsl:template match="from">
<p><b>De:</b> <xsl:value-of select="."/></p>
</xsl:template>
<xsl:template match="to">
<p><b>A:</b> <xsl:value-of select="."/></p>
</xsl:template>
<xsl:template match="subject">
<p><b>Tema:</b> <xsl:value-of select="."/></p>
</xsl:template>
<xsl:template match="message">
<p><b>Missatge:</b></p>
<p><xsl:value-of select="."/></p>
</xsl:template>
</xsl:stylesheet>

```

- Resultat HTML (`mail.html`)

```

<html>
<head>
<title>Mail</title>
</head>
<body>
<p><b>De:</b> pep@uoc.edu</p>
<p><b>A:</b> maria@uoc.edu</p>
<p><b>Tema:</b> Hola</p>
<p><b>Missatge:</b></p>
<p> Bon dia i bona hora.</p>
</body>
</html>

```

Recordeu que l'especificació XSLT és un llenguatge basat en XML que permet de definir un conjunt de regles per transformar els elements d'un document XML.

L'XSLT utilitza l'especificació XPath per a determinar els elements del document XML als quals cal aplicar cada regla. 

Una regla es defineix en XSLT mitjançant l'etiqueta `template`:

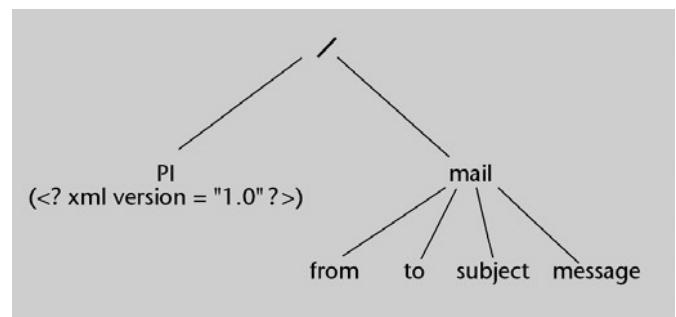
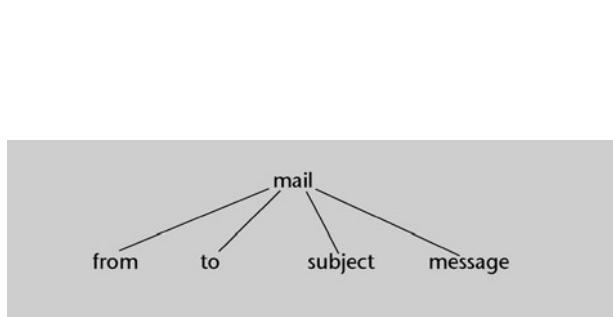
```
<template match="from">
  <p><b>De:</b> <value-of select="."/></p>
</template>
```

En aquest cas, quan es processa el node `from` de l'arbre XML, s'executa la regla que hi ha definida per a aquest element (la que teniu sobre aquestes línies).

Una regla `template` molt important és:

```
<template match="/">
...
</template>
```

Aquesta regla apareix en la majoria de documents XSLT i és la primera regla que el processador executa. L'element "/" representa tot el document XML, **inclosa la capçalera**, el qual es pot veure com l'autèntica arrel de l'arbre XML. Això és així per a permetre que els processadors també puguin tractar els elements que hi ha a la capçalera del document (per exemple, es podria voler recórrer les instruccions de procés o els comentaris que són elements que poden ser a la capçalera). Vegem-ho creant l'arbre XML de l'exemple del correu:

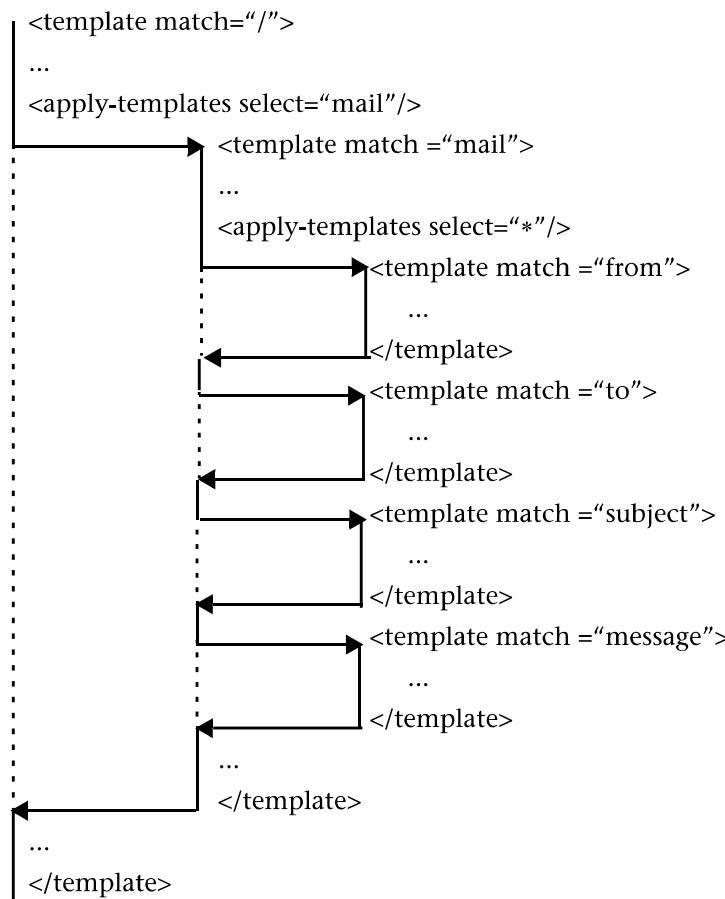


L'etiqueta `apply-templates` permet d'imposar restriccions sobre els nodes de l'arbre XML que es volen recórrer:

- `<apply-templates select="*"/>` indica que volem recórrer totes les etiquetes contingudes en l'etiqueta actual.
- `<apply-templates select="from"/>` indica que només volem recórrer les etiquetes `from` contingudes en l'etiqueta actual.

El contingut de l'atribut `select` de les etiquetes `template` i `apply-templates` és una expressió **XPath**.

En l'exemple anterior, les regles es van activant a mesura que ens anem desplaçant pels nodes de l'arbre XML. En el diagrama següent s'observen les activacions de regles que es produeixen:



Elements i funcions XSLT

Tot seguit es presenta un quadre resum dels principals elements que constitueixen l'especificació XSLT:

Element XSLT	Descripció
<ul style="list-style-type: none"> • stylesheet 	<p>És l'element arrel de l'especificació XSLT. És recomanable que tots els elements XSLT estiguin associats a un espai de noms i és en aquest element on es defineix aquest espai de noms. Per convenció, el prefix que es fa servir a aquest efecte és "xsl".</p> <p>Exemple:</p> <pre><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> ... </xsl:stylesheet></pre>
<ul style="list-style-type: none"> • output 	<p>Permet de definir propietats sobre el tipus de sortida que es vol generar com ara la codificació, el format de sortida, etc.</p> <p>Exemple:</p> <pre><xsl:output method="html"/></pre> <p>Indica que la sortida serà HTML</p> <pre><xsl:output method="xml" encoding="ISO-8859-1"/></pre> <p>Indica que la sortida és XML i que la codificació és ISO-8859-1. Això farà que al principi del document de sortida s'hi afegueixi automàticament la línia:</p> <pre><?xml version="1.0" encoding="ISO-8859-1"?></pre>

Element XSLT	Descripció
<ul style="list-style-type: none"> • template - param 	<p>Permet definir les regles a aplicar sobre els elements de l'arbre XML d'entrada. Pot tenir paràmetres d'entrada que s'especifiquen amb l'element param.</p> <p>Exemple:</p> <pre><xsl:template match="p"> Estem tractant un paràgraf. </xsl:template></pre>
<ul style="list-style-type: none"> • apply-templates 	<p>Permet d'activar els <i>templates</i> que fan "match" amb els nodes seleccionats amb l'expressió XPath de l'atribut <i>select</i>.</p> <p>Exemple:</p> <pre><xsl:apply-templates select="*"/></pre>
<ul style="list-style-type: none"> • call-templates - with-param 	<p>Permet de cridar un <i>template</i>. Els paràmetres de la crida s'especifiquen amb <i>with-param</i>.</p> <p>Exemple:</p> <pre><xsl:call-template name="aux"> <xsl:with-param name="param1">10</xsl:with-param> <xsl:with-param name="param2">20</xsl:with-param> <xsl:call-template></pre>
<ul style="list-style-type: none"> • for-each 	<p>Iterador. Permet de fer un recorregut per tots els nodes seleccionats per l'expressió XPath de l'atribut <i>select</i>.</p> <p>Exemple:</p> <pre><xsl:for-each select="*"> Element número: <xsl:value-of select="position()"/> </xsl:for-each></pre>
<ul style="list-style-type: none"> • if 	<p>Estructura condicional simple.</p> <p>Exemple:</p> <pre><xsl:if test="llibre/autor"> Autor: <xsl:value-of select="llibre/autor"/> </xsl:if></pre> <p>Només s'escriu el nom de l'autor si existeix.</p>
<ul style="list-style-type: none"> • choose - when - otherwise 	<p>Condicional múltiple. Estructura per a expressar múltiples condicions.</p> <p>Exemple:</p> <pre><xsl:choose> <xsl:when test="position()=3">La posició és 3</xsl:when> <xsl:when test="position()=4">La posició és 4</xsl:when> <xsl:otherwise>la posició no és ni 3 ni 4</xsl:otherwise> </xsl:choose></pre>
<ul style="list-style-type: none"> • value-of 	<p>Escriu el contingut (sense les marques, si n'hi ha) de l'expressió XPath seleccionada amb l'atribut <i>select</i>.</p> <p>Exemple:</p> <pre><xsl:value-of select="llibre/capitol[1]"/></pre>
<ul style="list-style-type: none"> • copy-of • copy 	<p>Crean una còpia del node seleccionat. Si s'utilitza <i>copy</i> es copia el node sense atributs ni elements fills, mentre que si s'utilitza <i>copy-of</i> es copia el node amb atributs i elements fills inclosos:</p> <p>Exemple:</p> <pre><xsl:copy select="."/> <xsl:copy-of select="."/></pre>
<ul style="list-style-type: none"> • text 	<p>Permet d'escriure text de manera literal a la sortida.</p> <p>Exemple:</p> <pre><xsl:text> </xsl:text></pre> <p>Escriu tres espais en blanc a la sortida.</p>
<ul style="list-style-type: none"> • variable 	<p>Permet de definir una constant.</p> <p>Exemple:</p> <pre><xsl:variable name="nom">Maria</xsl:variable> <xsl:value-of select="\$nom"/></pre> <p>escrivim a la sortida el valor de la constant <i>nom</i>. En aquest cas el valor és Maria.</p>

Element XSLT	Descripció
• include • import	Permet d'incloure les regles d'un full d'estil XSLT dins d'un altre. Exemple: <xsl:include name="compis.xslt"/>
• element	Permet d'afegir un nou element a l'arbre de resultats. Exemple: <xsl:element name="p"> Això és un nou paràgraf. </xsl:element>
• attribute	Afegeix un atribut a un element. Exemple: <xsl:element name="img"> <xsl:attribute name="src">imatge.gif</xsl:attribute> <xsl:attribute name="alt">Una imatge</xsl:attribute> </xsl:element>
• sort	Permet d'ordenar els nodes segons els valors d'un element o un atribut. Exemple: <xsl:for-each select="*" <xsl:sort select="@name"/> <xsl:copy-of "."/> </xsl:for-each> Ordena els nodes segons el valor de l'atribut <i>name</i> .

XSLT també disposa d'algunes funcions que poden ser d'utilitat. En destaquem les més importants:

Funció	Descripció
• current()	Fa referència al node que es tracta de l'arbre XML. Exemple: <xsl:value-of select="current()"/>
• document(string)	Permet d'accendir a un document XML extern. Exemple: <xsl:value-of select="document('llibre.xml')/llibre/titol"/>

Desenvolupament de documents XSLT

L'XSLT no és un llenguatge de programació sinó un llenguatge de transformació. A efectes pràctics, però, moltes de les tècniques de programació són perfectament vàlides per a escriure fulls d'estil XSLT. En general un programador se sent còmode quan escriu documents XSLT.

Com a metodologia de desenvolupament d'XSLT és molt aconsellable aplicar les tècniques del **disseny descendent** i del **disseny modular**. En aquest sentit, l'element **call-template** facilita enormement el disseny descendent i els elements **include** i **import** permeten la modularització. 

Podeu trobar un exemple complet de full d'estil XSLT en l'aula de l'assignatura. Es tracta d'un calendari que informa dels diferents esdeveniments programats en dates concretes.

WEB

Tot seguit es recullen un conjunt de consideracions que us poden ser d'utilitat en el desenvolupament de documents XSLT:

Recomanació	Intentar evitar l'ús de l'operador //
Problema	<pre><xsl:apply-templates select="//apartat"/></pre> <p>Aquesta instrucció busca per tot l'arbre XML qualsevol etiqueta amb nom "apartat" i requereix que el processador analitzi tots i cada un dels nodes de l'arbre, cosa que en fa disminuir molt el rendiment.</p>
Solució	<p>Si es coneix la ubicació exacta de l'etiqueta és molt millor accedir-hi directament, atès que es restringeix molt més les zones de l'arbre en què s'han de fer les cerques:</p> <pre><xsl:apply-template select= "/doc/capitol/modul/apartat"/></pre>

Recomanació	L'ús de variables pot simplificar molt el nombre de cerques que es fan en un XSLT
Problema	<p>Hi ha moltes cerques del mateix element:</p> <pre><xsl:value-of select="/agenda/amics/persona[name='Anna']"/> ... <xsl:value-of select="/agenda/amics/persona[name='Anna']"/> ... <xsl:value-of select="/agenda/amics/persona[name='Anna']"/></pre>
Solució	<p>Es pot definir variables per a les cerques més comunes d'un XSLT i consultar simplement el valor de la variable quan calgui fer la cerca. D'aquesta manera la cerca només es fa quan es crea la variable:</p> <pre><!--moment en què es fa la cerca --> <xsl:variable name="pAnna" select="/agenda/amics/persona[name='Anna'] "/> <!-- ara la cerca ja està feta i simplement es copia el valor emmagatzemat dins la variable --> <xsl:value-of select="\$pAnna"/> ... <xsl:value-of select="\$pAnna"/> ... <xsl:value-of select="\$pAnna"/></pre>

Recomanació	La classificació <xsl:sort .../> pot portar problemes de rendiment; si es pot evitar millor
Problema	Ordenar sempre comporta un cost temporal important.
Solució	Procurar que, en el document XML, les dades ja estiguin ordenades.

Recomanació	Disseny descendant
Problema	Com passa amb els programes, crear un full d'estil complex, requereix aplicar mètodes fiables que n'assegurin la qualitat i la mantenibilitat.
Solució	La tècnica que més s'aproxima a aquest propòsit és el disseny descendant. L'element <code>call-template</code> permet desenvolupar fulls d'estil aplicant disseny descendant.

Recomanació	Disseny modular
Problema	En tot procés complex, i la creació de fulls d'estil XSLT ho és, és molt important aïllar els problemes d'un projecte concret i fer extensibles les solucions a altres projectes similars. La reutilització és bàsica per a abaratir els costos i afavorir el manteniment. El disseny modular és una de les principals tècniques per a aconseguir-ho.
Solució	En el desenvolupament de fulls d'estil XSLT, el disseny modular es pot dur a terme gràcies als elements import i include, que permeten de reutilitzar altres fulls d'estil.

Recomanació	Tenir en compte les precedències entre regles i aprofitar-se'n
Problema	XSLT permet de crear regles (<i>templates</i>) i variables iguals i estableix tot un seguit de precedències entre si.
Solució	Treballar amb regles amb diferents precedències pot facilitar força la feina. Per exemple, en un document XSLT es podria tenir els dos <i>templates</i> següents: <pre><xsl:template match="para"> <p><xsl:copy-of select="*"/></p> </xsl:template> <xsl:template match="*" <copy-of select="."/> </xsl:template></pre> <p>En aquests casos, la regla superior té precedència sobre la inferior: quan es tracta un element <i>para</i>, s'activa la primera regla i se substitueix l'etiqueta <i>para</i> per <i>p</i>. Per a qualsevol altre element, la regla que s'activa és la inferior.</p>

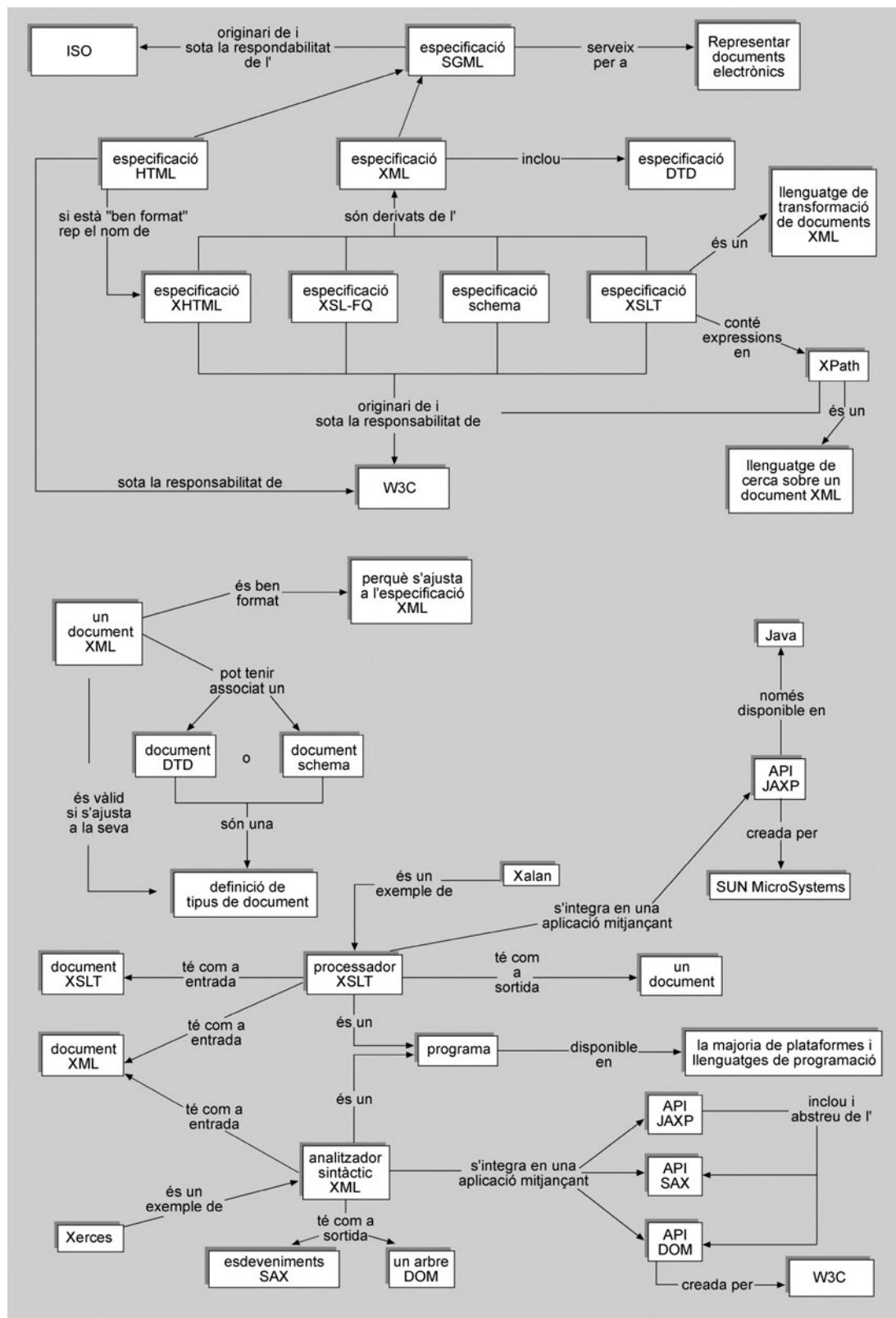
Recomanació	Programació recursiva
Problema	Sovint, en el desenvolupament de fulls d'estil XSLT, cal repetir de manera iterativa regles o parts de regles.
Solució	Hi ha una estructura iterativa, <i>for-each</i> , per a aquest propòsit, però malauradament no és gaire potent. Moltes vegades cal utilitzar tècniques de programació recursiva (l'element <i>call-template</i> permet de fer-ho). Es tracta de crear <i>templates</i> recursius.

Recomanació	Simplificar les expressions Xpath
Problema	Les expressions XPath poden ser una de les principals causes d'ineficiència.
Solució	Cal mirar de simplificar al màxim les expressions XPath i pensar en el cost temporal que poden tenir.

Recomanació	XSLT per a filtrar / XSLT per a donar format
Problema	Bàsicament els fulls d'estil XSLT permeten d'aplicar dues operacions sobre l'arbre XML d'entrada:
	<ul style="list-style-type: none"> • Filtrar, que consisteix a eliminar branques de l'arbre o afegir-n'hi • Donar format, que consisteix a crear una presentació per als elements de l'arbre.
Solució	Una bona tècnica és mirar de separar aquestes dues operacions en diferents fulls d'estil XSLT. Així, es disposarà d'un full d'estil de filtratge i d'un altre de presentació. Els dos fulls d'estil s'executen "en sèrie" (l'un després de l'altre). Aquesta tècnica permet de crear una altra capa lògica (separació de filtratge i presentació), que facilita encara més el desenvolupament.

Annex 4

Mapa d'estàndards



Glossari

acció semàntica *f* Conjunt d'accions associades a una regla sintàctica. Especifiquen com prenen valor els atributs associats a cada símbol.

analitzador lèxic *m* Algorisme o programa que fa l'anàlisi lèxica.

analitzador sintàctic *m* Algorisme o programa que fa l'anàlisi sintàctica.

analitzador sintàctic XML *m* Analitzador sintàctic del llenguatge XML.

analitzador *m* Vegeu analitzador sintàctic.

Apache.org. *f* Organització promotora de projectes de codi font obert i lliure distribuït molt popular a Internet. Ofereix una gran quantitat de productes orientats al desenvolupament d'aplicacions web.

API *f* Vegeu interfície d'aplicació del programa.

application programming interface *f* Vegeu interfície d'aplicació del programa.

arbre XML *m* Estructura de dades que representa els elements d'un document XML un cop aquest ha estat analitzat.

atribut heretat *m* Atribut d'un node de l'arbre sintàctic que obté el seu valor a partir dels atributs del node pare i dels nodes germans.

atribut sintàctic *m* Conjunt de valors que s'associen a un símbol de la gramàtica.

atribut sintetitzat *m* Atribut d'un node de l'arbre sintàctic que obté el seu valor a partir dels atributs dels nodes fills directes.

atribut XML *m* Parell nom, valor que expressa una propietat sobre un element XML.

BNF *m* Mètode formal de representació de gramàtiques de llenguatges.
en: *bakus-naur form*

BNF visual *m* Notació basada en BNF en què es busca facilitar al màxim la llegibilitat de la gramàtica.

CASE *f* Vegeu eina CASE.

computer-aided software engineering *f* Vegeu eina CASE.

CORBA *m* Estàndard de crides remotes a procediments i objectes adoptat per OMG.
en common object request broker architecture

CORBA IDF *m* Llenguatge de definició d'interfícies per a les classes i procediments CORBA.
en common object request broker architecture interface definition language

dada de caràcter *f* En un text marcat, tot allò que no són marques.

declaració de marcator *f* Notació descrita en l'especificació XML per a la representació de documents DTD.

disseny descendent *m* Tècnica de disseny que consisteix a descompondre un problema en subproblemes més senzills.

disseny modular *m* Tècnica de programació que consisteix a dividir un programa en subprogrames (anomenats *mòduls*). Cada mòdul duu a terme una tasca independent.

document *m* Porció d'informació significativa per a un usuari (home o màquina) concret.

document ben format *m* Document que satisfà la sintaxi de l'especificació XML.

document electrònic *m* Document que un ordinador pot emmagatzemar, manipular i presentar.

document type definition *f* Vegeu definició de tipus de document.

document XML vàlid *m* Document que s'ajusta a la seva DTD associada.

DOM *m* Jerarquia de classes que permeten de representar un document electrònic en memòria. Ofereix mètodes per a navegar i manipular l'estructura.
en document object model

DTD *f* Vegeu definició de tipus de document.

definició de tipus de document *f* Descripció en declaracions de marcatge d'un conjunt de regles XML per a la representació de documents XML d'un determinat tipus.
en document type definition
sigla: DTD

eina CASE *f* Programari de suport al desenvolupament, manteniment i documentació informatitzats de programari.
en computer-aided software engineering

element XML *m* Conjunt de components lògics que formen els documents XML.

esdeveniment *m* Fet que es pot produir en un instant en el temps i que pot provocar una transició o un canvi.
en event

espai de noms *m* Cadascun dels àmbits en què té un significat un element XML.

esquema XML *m* Llenguatge basat en XML que permet de definir una DTD.

etiqueta XML *f* Descriptor d'informació relativa a un element XML.

event *m* Vegeu esdeveniment.

expressió regular *f* Llenguatge per a especificar patrons que concorden amb una seqüència de caràcters.

extensible markup language *m* Llenguatge de marques generalitzat i extensible basat en SGML.
sigla: XML

extensible stylesheet language *m* Conjunt de les tres especificacions: XPath, XSL-FO i XSLT.
sigla: XSL

extensible stylesheet language - formatting objects *m* Llenguatge basat en XML per a especificar el format de presentació de text d'un document.
sigla: XSL-FO

extensible stylesheet language transformations *m* Llenguatge basat en XML que permet de definir un conjunt de regles per transformar els elements d'un document XML.
sigla: XSLT

fase d'anàlisi *f* Fase del compilador en què es fa l'anàlisi del fitxer font. La sortida sol ser la representació del fitxer en forma d'arbre sintàctic. Consta de l'anàlisi lèxica, sintàctica i semàntica.

formatting object *m* Vegeu objecte de format.

generador d'analitzadors lèxics *m* Programa que, a partir de les expressions regulars que defineixen els lexemes dels testimonis d'un llenguatge, genera el codi font d'un analitzador lèxic.

generador d'analitzadors sintàctics *m* Programa que, a partir de la descripció formal d'una gramàtica, genera el codi font d'un analitzador sintàctic.

generalized markup language *m* Vegeu llenguatge de marcatge generalitzat.

GML *m* Vegeu llenguatge de marcatge generalitzat.

gramàtica *f* Descripció formal de l'estructura sintàctica d'un llenguatge.

identificador universal de recursos *m* Seguit de normes per a especificar una referència unívoca a un recurs. Normalment consisteix en una forma ampliada de les URL (*uniform resource locator*) que s'utilitzen al Web.
en uniform resource identifier
sigla: URI

interfície *f* Conjunt d'operacions d'una classe visibles des d'altres classes.

interfície d'aplicació del programa *f* Conjunt de funcions, classes o mòduls que permeten d'interaccionar amb un programa.
en application programming interface
sigla: API

interfície d'usuari *f* Allò que veuen els usuaris del funcionament del programari.

JAXP *m* Vegeu Java API for XML processing.

Java API for XML processing *m* API de programació Java que ofereix una capa de connexivitat amb qualsevol analitzador XML i processador XSLT.
sigla: JAXP

JDBC *m* Capa de connexió amb bases de dades relacionals des de Java. Ofereix independència lògica respecte a la base de dades escollida.

JDOM *m* Jerarquia de classes similar a DOM però optimitzada pel seu ús amb el llenguatge de programació Java.

Lex *m* Programa generador d'analitzadors lèxicos.

lexema *m* Unitat lèxica mínima d'un llenguatge.

llenguatge *m* Conjunt de cadenes d'un alfabet construït segons les regles d'una gramàtica.

llenguatge de marcantge generalitzat *m* Llenguatge creat per IBM el 1969.
en generalized markup language
sigla: GML

llenguatge de marques *m* Llenguatge que descriu com s'ha d'intercalar un conjunt d'indicacions anomenades *marques* enmig d'un conjunt de dades de caràcter.

llenguatge de modelatge estàndard *m* Llenguatge per a la construcció de programari orientat a objectes.
en unified modeling language
sigla: UML

llenguatge de programació *m* Llenguatge formal en què estan escrits els programes d'ordinador.

llenguatge estandarditzat i generalitzat de marcantge *m* Norma internacional ISO 8879 per al marcantge de documents electrònics.
en standard generalized markup language
sigla: SGML

localitzador uniforme de recursos *m* Sistema universal de designació i localització d'un recurs a Internet (protocol, hoste, directori local). S'utilitza més en el sentit d'adreça d'una pàgina web.
en uniform resources locator
sigla: URL

marca *f* Instrucció d'un llenguatge de marques.

marcantge de format *m* Tipus de notació tipogràfica que permet d'indicar qui ha de ser el format d'una porció de text.

marcantge generalitzat *m* Marcantge en què cada una de les marques té un significat lògic.

model-view-controller *m* Model que és una arquitectura basada en tres tipus d'objectes i que permet d'obtenir un alt nivell de reutilització.
sigla: MVC

namespace *m* Vegeu espai de noms.

no terminal *adj* Dit del tipus de símbol utilitzat en les regles de la gramàtica que defineix un llenguatge. És el nom d'un element del llenguatge i denota un grup de símbols terminals.
Vegeu terminal.

objecte de format *m* Conjunt d'elements del llenguatge XML-FO.
en formatting object

Object Management Group *m* Organització no oficial d'estandardització de tot allò relacionat amb la programació orientada a objectes. Entre els principals estàndards aprovats desata l'UML com a llenguatge gràfic per a l'especificació de models orientats a objectes.
sigla: OMG

OMG *m* Vegeu Object Management Group.

parser *m* Vegeu analitzador sintàctic.

patró *m* Definició d'un conjunt de seqüències de caràcters que tenen unes característiques comunes.

precedència *f* L'ordre en què es fa una determinada operació en una expressió.

presentació *f* Resultat de convertir una *reproducció* en una cosa perceptible per a l'ésser humà.

processador de llenguatges *m* Programa d'ordinador que accepta un llenguatge d'entrada, el processa i el tradueix a un llenguatge de sortida.

processador de textos *m* Programari per a l'edició de documents electrònics.

processador XSLT *m* Eina de traducció de documents XML segons les indicacions d'un document XSLT.

programació recursiva *f* Mètode de programació que consisteix a aconseguir que un codi es repeteixi un nombre determinat de vegades fent que una funció es cridi a si mateixa.

recorregut en postordre *m* Lectura d'una estructura d'arbre en ordre esquerra, dreta, arrel.

regla de producció *f* Vegeu gramàtica.

regla sintàctica *f* Vegeu gramàtica.

reproducció *f* Combinació de dades i indicacions de format.

SAX *f* Vegeu simple API for XML.

simple API for XML *f* API per a l'anàlisi de documents XML basada en esdeveniments programables.

sigla: SAX

sentència *f* Cadena que pertany a un llenguatge i forma una construcció amb un significat sintàctic mínim.

serialització *f* Procés pel qual es grava en disc una estructura que està en memòria.

SGML *m* Vegeu llenguatge estandarditzat i generalitzat de marcatge.

standard generalized markup language *m* Vegeu llenguatge estandarditzat i generalitzat de marcatge.

símbol *m* Nom que apareix en una producció d'una gramàtica.

sintaxi *f* Estructura de les cadenes d'un llenguatge.

tag *f* Vegeu marca.

terminal *m* Tipus de símbol que pot aparèixer en les sentències d'un llenguatge.

text XML *m* Conjunt de dades de caràcter i marques.

token *m* Element individual d'una sentència.

traductor *m* Programa que llegeix un codi escrit en un llenguatge font i el converteix en un codi escrit en un llenguatge objecte.

UML *m* Vegeu lleguatge de modelatge estàndard.

unified modeling language *m* Vegeu llenguatge de modelatge estàndard.

uniform resource identifier *m* Vegeu identificador universal de recursos.

uniform resources locator *m* Vegeu localitzador uniforme de recursos.

URI *m* Vegeu identificador universal de recursos.

URL *m* Vegeu localitzador uniforme de recursos.

W3C *m* Vegeu World Wide Web Consortium.

World Wide Web Consortium *m* Consorci d'institucions comercials i educatives que s'encarrega de promoure els estàndards a tots els camps relacionats amb el WWW.
sigla: W3C

what you see is what you get *loc* Manera d'anomenar aquells programes de processament de textos en què la interfície d'usuari per a la reproducció del document està dissenyada per a tenir l'aspecte d'una presentació.

sigla: WYSIWYG

WYSIWYG *m* Vegeu what you see is what you get.

XML *m* Vegeu extensible markup language.

XML schema *m* Vegeu esquema XML.

XPath *m* Llenguatge basat en expressions que permet de cercar informació emmagatzemada dins una estructura XML.

XSL *m* Vegeu extensible stylesheet language.

XSL-FO *m* Vegeu extensible stylesheet language - formatting objects.

YACC *m* Vegeu yet another compiler compiler.

yet another compiler compiler *loc* Programa generador d'analitzadors sintàctics. Vegeu generador d'analitzadors sintàctics.

sigla: YACC

Bibliografia

Aho, A.V.; Sethi, R.; Ullman, J.D. (1990). *Compiladores, principios, técnicas y herramientas*. Addison-Wesley Iberoamericana.

Burke, E.M. (2002). *Java and XSLT* [traducció castellana: *Java y XSLT*. Anaya]. O'Reilly.

Goldfarb, Ch.F.; Prescod, P. (1999). *XML Handbook*. Prentice Hall.

McLaughlin, B. (2001). *Java and XML* [traducció castellana: *Java y XML en la seva versió castellana*. Anaya]. O'Reilly.

Rusty Harold, E. (2001) *XML Bible* (2a. ed.). IDG Books Worldwide Inc.

