Université Mohammed V
Faculté des Sciences
Rabat

Pr. El Mamoun SOUIDI

Master IAO
Année 2022

Table des matières

Chapitre 1

Il était une fois SGML (Standard Generalized Markup Language ou langage de balisage standard généralisé ISO 8879 en 1986), le premier essai de standardisation de documents électroniques. Il a été utilisé pour la documentation administrative, technique, scientifique et autres. Mais il est très complexe, notamment pour être utilisé de façon standard sur le Web. XML est développé par le W3C [1] essentiellement pour dépasser les limites de HTML et la complexité de SGML. XML est à sa deuxième version :
- XML 1.0 en 1998, version très appréciée.
- XML 1.1 en 2004, version de mise à jour et en particulier la conformité avec l'Unicode [2].

Avantages de XML : interopérabilité, pérennité des documents XML, possibilité de donner plusieurs mise en forme à un même document XML (HTML, pdf, LaTeX, etc) en plus de la possibilité de faire des extractions d'information, des tris etc... XML est aussi utilisé efficacement pour l'échange de données entre applications.

La syntaxe de XML est simple. Il s'agit d'un ensemble de règles à respecter mais de façon très stricte.

1 Généralités sur XML

XML (eXtensible Markup Language) un méta-langage universel pour structurer les données. Un document XML est un fichier texte. XML est un langage de balisage. Il permet de

1. Consortium de développement des langages du Web, `http://www.w3.org`
2. Un standard de codage sur 16 bits, de presque tous les caractères des langues dans monde. `www.unicode.org`

marquer (baliser) les documents afin de les structurer. C'est donc un langage de structuration de document. En XML, les balises ne sont pas prédéfinies : tout un chacun peut définir ses propres balises.

XML est libre de droits, indépendant de toute plate-forme et correctement pris en charge. Ce n'est pas un format propriétaire comme Word ou PDF. Une application importante de XML est le langage XHTML comme successeur de HTML.

En XML, la mise en forme d'un document est complètement séparée du contenu de ce document. Ce qui permet de fournir plusieurs types de sortie pour un même fichier XML. Par exemple : un autre fichier XML, un fichier HTML, un tableau, un graphique, image, animation multimédia, fichier PDF...). XML permet aussi d'effectuer des extractions, des sélections par tri, des générations automatiques de tables des matières et bien d'autres fonctions, grâce au langage de feuilles de style XSLT (eXtensive Stylesheet Language Transformations) à voir plus tard. Le XML est aussi utilisé pour l'échange de données entre applications.

Actuellement, toute application professionnelle doit avoir comme sortie un document XML pour pouvoir le transformer, par la suite, en d'autres formats. XML est important à savoir et facile à apprendre.

XML structure l'information à stocker et à échanger entres différentes applications. On le trouve aujourd'hui dans tous les domaines :

- édition (description d'ouvrage avec DocBook),

- graphisme (format SVG),

- mathématiques (formules avec MathML),

- chimie (CML permet la description de molécules en 3D),

- musique (partition musicale avec MusicML), etc.

Un document XML à lui seul ne fait rien. Il est accompagné de deux autres :

- un fichier XSLT contenant les informations de transformation et de mise en forme, il permet de produire le format de sortie voulu ;

- un fichier DTD (Document Type Definition), ou un Schema XML optionnel pour définir a priori les balises et leurs imbrications et occurrences, les attributs et leur valeurs, etc. Ce qui faciliterai la coordination en cas ou plusieurs personnes travaillent sur ces fichiers XML.

Pour la mise en forme de document XML, on peut aussi utiliser le langage CSS et les navigateurs.

XML est extensible et rigoureux, comme on va le voir. Cette rigueur de XML (sensibilité à la casse, guillemets obligatoires etc.) permet des traitements automatiques. Un document XML, est un fichier texte, peut facilement être modifié par les humains.

2 Le langage XML

En XML chacun définit ses balises. XML est sensible à la casse. Il vaut mieux n'écrire les balises qu'en minuscules.

En XML, les commentaires se déclarent comme en HTML et SGML par <!- - et se terminent par - ->. La chaîne de caractères - - est alors interdite dans un commentaire. Aussi, un commentaire dans un élément est interdit. L'exemple ci-dessous est incorrect

```
<produit nom="DVD"
prix='100' <!-- en Dhs -->
/>
```

2.1 Élément et Balise en XML

Une balise est une chaîne de caractère de la forme <x> ... </x> ou simplement <x/> qui permet de marquer ou séparer les différentes partie d'un texte.

Par exemple : Le numéro de téléphone de M. <nom> Ali</nom> âgé de <age>26</age> ans est <tel>0666666666</tel> ...

La balise de début, la balise de terminaison et le contenu sont globalement appelés élément. Ainsi, un élément XML est composé d'une balise ouvrante : <x>, d'un contenu, et d'une balise fermante </x>.

En XML, les balises ne sont pas prédéfinies. Chacun définit ses propres balises. En respectant les règles de nomination suivantes :

1– Seuls les lettres, les chiffres, tiret, tiret bas, et le tréma " :" sont autorisés dans les noms de balises en XML. (donc, ni espace ni symbole spécial comme $, # etc, ni symbole de ponctuation)

2– Un nom de balise commence nécessairement par une lettre et ne peut commencer par un chiffre, tiret, tiret bas, et le tréma " :" ou les lettres xml (quelque soit la casse). Le nom peut ensuite être composé de lettres, chiffres, tirets, tirets bas et tréma.

La longueur des noms est libre mais, il faut rester raisonnable et le plus explicitement possible.

Les caractères accentués francophones comme é, à, ê, ï, ù etc... sont théoriquement permis mais pourraient être mal interprétés par certaines applications traitant les fichiers XML. Donc à éviter. Il en est de même pour les caractères d'autres langues (arabe, ...) avec la version XML 1.1..

Toutes les balises portant un contenu non vide doivent être fermées. Par exemple <x> ...</x>. Les balises n'ayant pas de contenu doivent se terminer par />. Par exemple <x/>.

L'imbrication de balise doit être correctement faite. Ainsi `<x>...<y>...</x>...</y>` est faut. Par contre `<x>...<y>...</y>...</x>` est correcte.

2.2 Attribut

Un attribut est une information supplémentaire attachée à un élément. Un élément peut avoir ou non des attributs. Ceux-ci sont toujours signalés dans la balise d'ouverture. Un attribut est composé d'un nom et d'une valeur entre guillemets ou apostrophes. Les règles de nomination de balise s'appliquent aux attributs.

La syntaxe est

```
<balise attribut1="valeur1" attribut2="valeur2" ...> ... </balise>
```

Tout attribut doit avoir une valeur. La valeur doit être entourée de guillemets doubles ou d'apostrophes. Si une valeur contient des apostrophes, elle doit être entourée de guillemets et inversement.

Un élément ne peut contenir qu'une seule fois le même attribut. L'ordre des attributs est sans importance.

Exemple d'attributs prédéfinis en XML :

`xml:lang` permet de spécifier la langue du contenu de l'élément. Les valeurs possibles sont les codes pour la présentation des langues selon la norme ISO 639. Par exemple : `ar` pour l'arabe, `fr` pour le français, `en` pour l'anglais etc...

```
<remerciement xml:lang="fr"> merci </remerciement>
<remerciement xml:lang="en"> thank you </remerciement>
```

Un autre attribut prédéfini en XML est `xml:space`. Il spécifie si les espaces doivent être préservés dans le contenu de l'élément. Il a deux valeurs possibles : la valeur par défaut est `default` et `preserve` qui veut dire que les caractères de séparation seront préservés.

2.3 Structure d'un document XML

Un fichier XML est composé d'un prologue, d'un élément racine (unique) et d'un arbre. Cet arbre est constitué d'éléments imbriqués correctement les uns dans les autres (ayant une relation parent-enfant) et d'éléments frères dit aussi adjacents.

Le prologue est la déclaration XML suivante :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

Elle doit être placée en toute première ligne du document XML, ne doit être précédée ni par commentaire, ni même par une ligne vide. Les attributs `version`, `encoding` et `standalone` doivent être placés dans cet ordre.

1. `version` : version du XML utilisée dans le document, 1.0 de 1998 ou 1.1 de 2004 pour l'utilisation de l'Unicode ;

2. `encoding` : l'encodage de caractères utilisés. Sont autorisés les jeux de caractères définis dans la norme ISO/IEC-10646. Par exemple

`encoding="UTF-8"` jeu de caractères international de largeur de 8 bits par caractère.
`encoding="UTF-16"` jeu de caractères international de largeur de 16 bits par caractère.
`encoding="ISO-8859-1"` pour les langues de l'Europe de l'ouest.
`encoding="ISO-8859-2"` pour les langues de l'Europe de l'est.
`encoding="ISO-8859-3"` pour les langues de l'Europe du sud.
`encoding="ISO-8859-4"` pour les langues de l'Europe du nord.
`encoding="ISO-8859-5"` pour les langues cyrilliques.
`encoding="ISO-8859-6"` pour les langues arabes. etc

3. `standalone` (facultatif) : dépendance du document vis à vis d'autres fichiers, comme DTD, schema XML, feuille de XSLT etc. Si `standalone="yes"`, le processeur de l'application n'attend aucun autre fichier. Sinon, le processeur attend un document externe. La valeur par défaut est `"yes"`.

- L'élément racine est unique et englobe tous les autres éléments. Il s'ouvre juste après le prologue, et se ferme à la fin du document.

- Les éléments : peuvent contenir du texte, ou bien d'autres éléments, en format d'arborescence.

Exemple

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>
<biblio>
      <livre isbn="2100069349">
            <auteur> ED TITTEL </auteur>
            <titre> XML </titre>
            <editeur> EdiSciences </editeur>
      </livre>
Ici du texte non inclue dans aucune balise.
      <livre isbn="2841772225">
            <auteur> AEleen Frisch </auteur>
            <titre> Les bases de l'administration système </titre>
            <editeur> O'Reilly </editeur>
      </livre>
</biblio>
```

On dit qu'un document est formé s'il respecte rigoureusement la syntaxe XML.

2.4 Caractères spéciaux

Pour les caractères spéciaux ayant un sens précis en XML, il est nécessaire de leur trouver un remplaçant lorsque l'on a besoin de les insérer dans un document XML.

Caractère	Entité
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

2.5 Les sections CDATA

Une section CDATA permet de définir un bloc de caractères ne devant pas être analysés par les processeurs XML. Par exemple un code source.

Exemple

```
<![CDATA[ Une balise commence par un < et se termine par un >.]]>
```

3 Mise en forme de document XML

Le langage CSS (Cascading Style Sheets) permet de donner de la mise en forme aux documents XML en utilisant les navigateurs.

Pour relier un fichier XML à un fichier CSS nous utilisons la syntaxe :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<?xml-stylesheet type="text/css" href="chemin/fichier.css"?>
<racine>
...
<racine>
```

Pour gérer les retours à la ligne en XML, on utilise la propriété `display` de CSS qui a plusieurs valeurs :

`block` : l'élément est affiché seul sur une ligne.

`inline` : valeur par défaut, l'élément est affiché sur la ligne.

`none` : empêche l'affichage de l'élément et de ses descendants.

Exemple : on place les fichiers XML et CSS suivants dans un même répertoire et on ouvre le fichier .xml par un navigateur.

document `.xml` :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<?xml-stylesheet href="fichier.css" type="text/css"?>
<bibliotheque>
      <livre>
            <!-- Élément enfant titre -->
            <titre>Les Misérables</titre>
            <auteur>Victor Hugo</auteur>
            <editeur> Masson </editeur>
            <nb_tomes>3</nb_tomes>
      </livre>
      <livre>
            <titre>L'Assomoir</titre>
            <auteur>Émile Zola</auteur>
            <editeur> ... </editeur>
      </livre>
      <livre lang="en">
            <titre>David Copperfield</titre>
            <auteur>Charles Dickens</auteur>
            <editeur> ... </editeur>
```

```
      <nb_tomes>3</nb_tomes>
      </livre>
</bibliotheque>
```

Fichier CSS : `␣fichier.css`

```
bibliotheque, livre {}
auteur {display: block;
        width: 250px;
        font-size: 16pt ;
        font-family: arial ;
        font-weight: bold;
        background-color: pink;
        color: white;
        padding-left: 10px;
            }
titre {display: block;
        font-size: 12pt;
        padding-left: 10px;
            }
editeur {
        display: block;
        font-size: 12pt;
        color: red ;
        font-weight: bold;
        padding-left: 10px;
            }
isbn {
        display: block;
       font-size: 11pt ;
       font-style: italic;
       font-family: arial ;
       padding-left: 10px;
            }
```

4 Outillage XML

Tout éditeur de texte peut être utilisé pour écrire des documents XML. Cependant, il existe des éditeurs spécialisés offrant plus de fonctionnalités comme vérifier dynamiquement que le fichier est bien formé, vérifier dynamiquement la validité vis à vis de schéma, proposer des fonctions d'auto-complétion, etc

Par exemples :

– Jaxe, libre et multi-plateformes (http ://jaxe.sourceforge.net/fr/)

– xmlcopyeditor sous Linux libre et gratuit permet la validation aussi

– oXygen (http ://www.oxygenxml.com/) commercial complet, multi-plateformes (Linux, Mac, Windows) ;

– XML Spy, disponible uniquement sous Windows ;

– Scenari, chaîne éditoriale complète libre et multi-plateformes (http ://scenari-platform.org/)

5 Galaxie XML

Autour de XML on trouve plusieurs langages :

- XML : la syntaxe de base des balises

- XSL (eXtensible Stylesheet Language) langage évolué pour la définition de feuilles de style pour XML, composé de :

  - XSLT : transformations de document XML

  - XSLT-FO : formatage en vue de l'affichage

- XPath : mécanisme pour localiser toute partie d'un document XML.

- Espaces de noms : éviter les conflits entre noms de balises pour différentes applications

- Xlink, décrit une méthode standard pour ajouter des liens hypertextes à un fichier XML.

- Schema XML : définir la syntaxe des éléments, plus riche, plus précise et écrite en XML.

- XPointer et XFragments sont des syntaxes pour pointer sur des parties d'un document XML. XPointer ressemble à un URL, mais au lieu de pointer sur des documents du Web, il pointe sur des éléments de données au sein d'un fichier XML.

- XML Query : langage de requêtes.

- XForms : langage de définition des données pour des formulaires.

Chapitre 2

Rappelons qu'en XML chacun peut définir ses propres éléments et attributs. Dans un contexte professionnel où il y a plusieurs intervenants, il est nécessaires de coordonner la structure de documents XML. D'où la nécessité d'établir des règles, appelées schéma, définissant les éléments ainsi que leur imbrication et occurrence, les attributs et plus généralement la structure de documents XML.

Un schéma est une description de la structure que doit respecter un document XML lui faisant référence. On parle aussi de grammaire, au sens où le schéma définit l'enchaînement autorisé des balises et vient en complément de la syntaxe XML, vue au chapitre précédant. Un document XML qui déclare un schéma doit s'y conformer.

Il y a plusieurs langages qui permettent de développer ces schémas ou règles : DTD héritée de l'époque SGML, XML Schema une recommandation de W3C, Relax NG (REgular LAnguage for XML Next Generation), Schematron et d'autres.

Ce chapitre traite le langage DTD. Il est simple mais limitée. Les schémas XML décrivent plus finement la structure de documents XML. C'est l'objet du prochain chapitre.

DTD définit les éléments, les sous éléments et les attributs autorisés dans un document XML ainsi que la hiérarchie des éléments et leur fréquence. DTD définit la grammaire pour construire les documents XML qu'elle valide. En particulier une DTD contient :
- l'arborescence des éléments et leur modèle d'imbrication ;
- l'ordre des éléments ;
- la fréquence d'apparition ;
- les attributs de chaque élément (nom, type et valeur par défaut) ;
- les entités binaires ou textuelles pouvant être incluses dans un document (les caractères

non ASCII par exemple) ;
- les notations qui servent à identifier les types spécifiques de données externes binaires.

Avantages DTD :
- faciliter la coopération, l'échange et la mise en commun de documents produits par des rédacteurs différents ;
- aide les développeurs d'applications pour traiter les documents XML respectant la même DTD ;
- les DTD sont réutilisables ;

Inconvénients :

- DTD n'impose pas de contraintes sur la forme et les types de contenus des éléments et des attributs ainsi que la fréquence d'apparitions d'un élément ne peut pas être contraint avec précision.

- Le langage DTD n'est pas un langage XML,

6 Généralités sur les DTD

DTD est un langage déclaratif ne dépendant pas de la casse. Une DTD est un ensemble de déclarations texte qu'on met dans le fichier XML ou dans un fichier externe qu'on rattache au fichier XML. Une DTD contient les déclarations : d'éléments, d'attributs, d'entités générales, et des commentaires, etc.

Le retour à la ligne est optionnelle, mais vivement conseillé pour la lisibilité du code. Les noms des éléments et des attributs doivent être des noms XML.

Les commentaires en DTD sont faits grâce à `<!-- ... -->` comme en HTML et XML.

La déclaration de la DTD est introduite par le mot clé DOCTYPE dont la syntaxe est

```
<!DOCTYPE racine ...>
```

où racine est la racine du document XML. La déclaration de DTD doit être juste après le prologue. La DTD peut être interne, externe ou mixte.

DTD interne

La syntaxe pour insérer une DTD dans un fichier XML est :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<!DOCTYPE racine[
... Contenu de la DTD ...
]> <!-- Fin DTD -->
<racine> <!-- racine du document XML -->
  ... Contenu du document XML ...
```

```
</racine>
```

Comme la DTD est interne, le fichier est indépendant d'où `standalone="yes"`.

DTD externe

La DTD est mise dans un fichier texte d'extension .dtd et relier au fichier XML comme il suit :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE racine SYSTEM "chemin/fichier.dtd">
<racine>
  ...
</racine>
```

`SYSTEM` : est utilisé pour donner l'adresse du fichier qui contient la DTD. Chemin peut être local ou un URL de la forme http ://www....

On peut aussi utiliser `PUBLIC` au lieu de `SYSTEM`. PUBLIC : est utilisé lorsque la DTD est une norme ou qu'elle est enregistrée sous forme de norme ISO : Par exemple XHTML :

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

DTD mixte

Il est possible de mélanger les deux DTD interne et externe dans une DTD mixte en utilisant la syntaxe :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<!DOCTYPE racine SYSTEM "chemin/fichier.dtd" [
... suite de la DTD ...
]>
<racine>
. . .
</racine>
```

7 Déclaration d'éléments

Les noms des éléments à définir doivent respecter les règles de nominations XML. Tout élément à utiliser dans XML doit être déclaré une seule fois dans la DTD.

La forme générale pour déclarer un élément dans une DTD est :

```
<!ELEMENT nomElement modèle_de_contenu >
```

où modèle de contenu peut être : `(\#PCDATA)`, `ANY` ou `EMPTY`.

– `(\#PCDATA)` : (signifie Parsed Character DATA, toujours entre parenthèses) utilisé pour un contenu purement textuel sans éléments. Il peut contenir des références d'entité telle que & , < , etc. Mais pas les symboles <, >, & etc.

Par exemple, dans la DTD de XHTML 1.0 l'élément title qui permet de donner le titre de la fenêtre est déclaré par :

```
<!ELEMENT title (#PCDATA)>
```

Une adresse email peut aussi être déclarée par

```
<!ELEMENT email (#PCDATA) >
```

– `ANY` : l'élément peut contenir n'importe quel contenu : textes ou éléments. Ce type va à l'encontre de l'esprit de DTD, qui vise à instaurer des contraintes.

Par exemple dans la DTD de HTML la balise p est définit par

```
<!ELEMENT p ANY>
```

– `EMPTY` : pour déclarer un élément vide qui ne contiendra ni texte, ni d'autres éléments. Un élément vide peut avoir ou non des attributs. Par exemple dans la DTD de HTML, les balises `<br>` et `<img>` sont définis par

```
<!ELEMENT br EMPTY>
<!ELEMENT img EMPTY>
```

En XML
</br> ou <br/> sont valides mais
 </br> ne l'est pas.

7.1 Déclaration de sous-éléments

Tous les éléments et sous-éléments doivent être déclarés pour les utiliser dans les documents XML. Pour définir elt1, elt2 ... comme sous éléments de elt et **dans cet ordre** d'apparition dans un document XML, on utilise la syntaxe :

```
<!ELEMENT elt(elt1, elt2?, elt3+, elt4*,...)>
```

Chaque sous-élément doit de plus être déclaré ailleurs, en tant qu'élément dans la DTD (avant ou après cette définition) pour spécifier son modèle de contenu `(\#PCDATA)`, `ANY` ou `EMPTY`.

De plus, chaque sous élément peut être suivi d'un des symboles suivants :

– Absence de symbole : le sous élément apparaît une et une seule fois dans l'élément.

– Un sous-élément suivi de **?** apparaîtra au plus une fois : zéro ou une seule fois dans XML.

– Un sous-élément suivi de **+** apparaîtra une ou plusieurs fois dans XML.

– Un sous-élément suivi de * : apparaîtra 0 ou plusieurs fois dans XML.

L'exemple suivant indique que l'élément identité doit contenir, dans cet ordre, un ou plusieurs éléments prénom, un élément surnom facultatif et exactement un élément nom :

```
<!ELEMENT identité (prénom+,surnom?,nom)>
```

Exemple de DTD bibliothèque.

```
<!ELEMENT bibliotheque (livre+)>
<!ELEMENT livre (titre, auteur, ref)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (nom, prenom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT ref (#PCDATA)>
```

L'ordre des sous éléments, (titre, auteur, ref) et (nom, prenom), dans le document XML doit être respecté. Par exemple, dans la DTD de XHTML 1.0, les sous-éléments head, body de html sont définis comme il suit :

```
<!ELEMENT html (head, body)>
```

7.2 Sous éléments exclusifs

Pour définir des sous-éléments qui peuvent apparaître de manière exclusive. On utilise le symbole `|`. La syntaxe est :

```
<!ELEMENT elt(elt1|elt2?|elt3*| ...)>!
```

De plus, en utilisant les symboles d'occurrence, on peut préciser le nombre d'apparition de l'élément choisi. Cette syntaxe déclare un élément elt et ses sous éléments elt1, elt2, ... L'élément elt dans un document XML pourra contenir soit elt1 une seule fois, soit contenir elt2 au plus une fois, soit contenir elt3* 0 ou plusieurs fois.

Exemples :

```
<!ELEMENT situation (celibataire | marie | divorce)>
```

```
<!ELEMENT pièce_identité (carte_nationale|passeport)>
```

pièce_identité contient soit l'élément carte_nationale soit passeport, mais pas les deux,

7.3 Élément à contenu mixte

Un élément à contenu mixte peut contenir aussi bien du texte ou une section `CDATA` que des sous-éléments. Cela se fait avec une alternative entre un contenu de type `(\#PCDATA)`, toujours en premier et des sous-éléments ; cette alternative pouvant se répéter plusieurs fois grâce au symbole *. La syntaxe est

```
<!ELEMENT elt (#PCDATA | elt1 | elt2 | ...)*>
```

Par exemple

```
<!ELEMENT citation (#PCDATA | auteur)*>
Exemple d'utilisation :
<citation>
 <auteur>Shakespeare</auteur>To be or not to be
</citation>
```

```
<!ELEMENT parcours (#PCDATA | diplôme)*>
```

Ici, on a un élément parcours qui a un contenu mixte : du texte et pouvant contenir un nombre quelconque de sous-éléments diplôme.

7.4 Regroupement de sous-éléments

Il est possible d'utiliser les parenthèses pour regrouper des déclarations. Le modèle de contenu est défini pour chaque élément. Par exemple :

```
<!ELEMENT EXEMPLE (A, B?, C*, D+, E|F)>
```

signifie qu'un élément Elt doit toujours contenir un élément A, suivi optionnellement de B, suivi par zéro ou plusieurs C, suivi par un ou plusieurs D, suivi par E ou bien F.

```
<!ELEMENT cercle (centre, (rayon | diametre))>
```

Il est possible de combiner les syntaxes vues précédemment à l'aide de parenthèses

```
<!ELEMENT centre((x,y)|(r,theta))>
```

Si l'ordre est sans importance

```
<!ELEMENT centre((x,y)|(y,x)|(r,theta)|(theta,r))>
```

Dans la DTD de XHTML 1.0, les éléments ul et dl sont déclarés par :

```
<!ELEMENT ul li+>
<!ELEMENT dl (dt|dd)+>
```

```
<!ELEMENT paragraphe (#PCDATA|nom|prénom|note|date)*>
```

paragraphe peut contenir n'importe quel nombre d'éléments nom, prénom, note, date et dans n'importe quel ordre.

8 Déclaration d'attributs

Tous les attributs d'éléments à utiliser dans un document XML doivent être déclarés dans une DTD. Comme le nom de tout élément, le nom de tout attribut doit être un nom XML. On ne peut pas spécifier l'ordre d'apparition des attributs dans un élément. Leur ordre est sans importance.

Un seul ATTLIST permet de définir plusieurs attributs pour un seul élément. Dans le cas où un même attribut est commun à plusieurs éléments, il doit être déclaré séparément pour chaque élément. Pour chaque attribut on déclare :

– son nom, (doit être un nom XML)

– son type, (voir la liste des types ci-dessous, il y en a dix)

– sa valeur par défaut (s'il y en a) d'un attribut doit être mise entre " " ou ' '.

– son caractère optionnel ou non

La syntaxe est :

```
<!ATTLIST nomElément nomAttribut typeAttribut valeurParDefaut
     optionnelOuNon>
```

8.1 Caractère optionnel ou non d'attributs

Chaque déclaration d'attribut est précisée par son caractère optionnel ou non. Un attribut peut faire l'objet de l'un des caractère suivants :

1) Requis `\#REQUIRED` : signifie que l'attribut doit être obligatoirement déclaré et sans valeur par défaut. (par exemple en HTML il est obligatoire de préciser l'attribut src de l'élément img).

2) Optionnel `\#IMPLIED` signifie que l'attribut est facultatif et n'a pas de valeur par défaut particulière. Par exemple, les attributs width et height de <img>.

Exemples :

```
<!ATTLIST image source  CDATA #REQUIRED
hauteur CDATA #IMPLIED
largeur CDATA #IMPLIED
alt CDATA #IMPLIED
>


<!ATTLIST meta http-equiv CDATA #IMPLIED
name CDATA #IMPLIED
content CDATA #REQUIRED
scheme CDATA #IMPLIED >
```

3) `\#FIXED␣"valeur"` ou `\#FIXED␣'valeur'` : l'attribut a cette valeur et celle-ci n'est pas modifiable. Il est facultatif, mais si il est présent, il ne peut prendre que "valeur" comme

valeur et si il est absent on considère que sa valeur est "valeur". Cette valeur doit être du type de l'attribut.

```
<!ATTLIST svg xmlns CDATA #FIXED "http://www.w3.org/2000/svg">
<!ATTLIST livre xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink">
```

Pour XHTML1.0, les attributs http-equiv, name, content, scheme de l'élément meta sont des chaînes de caractères :

8.2 Types d'attribut

Il y a dix types d'attributs :

CDATA pour désigner une chaîne de caractères quelconque. Il est même possible d'utiliser des caractères habituellement réservés à XML : <, > et &.

Attribut à valeurs énumérées La liste des valeurs possibles d'un attribut peut être limitée. À la place du type, il suffit de spécifier toutes les alternatives possibles comme valeur. Pour définir une valeur par défaut éventuelle, il suffit de faire suivre l'énumération par la valeur désirée entre guillemets doubles ou apostrophes. La syntaxe est

```
<!ATTLIST elt attr1 (val1 | val2 | ...) #REQUIRED
              attr2 (val3 | val4) "val3" >
```

Exemple de déclaration d'une liste de choix d'attributs :

```
<!ELEMENT img EMPTY>
<!ATTLIST img format (GIF | JPEG | PNG) "PNG">
```

Si aucune valeur n'est affectée à cet attribut, c'est la valeur par défaut qui le sera, ici PNG. On notera l'absence de guillemets dans la liste des valeurs possibles.

NMTOKEN nom obéissant aux règles de nomination d'élément XML mais sans restriction sur le premier caractère. Par exemple, un code postal, un numéro de téléphone etc pourront être déclarés de ce type :

```
<!ATTLIST elt attr NMTOKEN #REQUIRED>
< elt attr="5-22-06-13-01-022"/>
```

Par exemple, un code postal pourra être déclaré de ce type :

```
<!ATTLIST ville nom  CDATA    #REQUIRED
                code NMTOKEN  #REQUIRED>
```

NMTOKENS Une suite de NMTOKEN, séparés par des espaces.

```
<!ATTLIST article no NMTOKENS #REQUIRED>
<article no="1234 14 1234567"/>
```

ID : (identifiant) la valeur d'un attribut de type ID doit être un nom XML et unique dans tout le document XML. Un élément peut avoir un seul attribut de ce type. Un attribut de type ID permet d'identifier de façon unique un élément du document.

```
<!ATTLIST diplôme intitulé CDATA #REQUIRED
          code ID #REQUIRED>
```

IDREF : la valeur d'un attribut IDREF doit correspondre à la valeur d'un attribut ID dans le document en cours.

```
<!ATTLIST elt att IDREF #REQUIRED>
```

IDREFS : des références vers plusieurs identifiants du document. Une suite de IDREF, séparées par des espaces.

```
<!ELEMENT elt1 (elt2 | elt3 | elt4)*>
<!ELEMENT elt2 (#PCDATA)>
<!ELEMENT elt3 (#PCDATA)>
<!ELEMENT elt4 (#PCDATA)>
<!ATTLIST elt2 id ID #REQUIRED>
<!ATTLIST elt3 ref IDREF #IMPLIED>
<!ATTLIST elt4 refs IDREFS #IMPLIED>
```

Le langage XLink généralise ce mécanisme de création de liens entre parties d'un même document.

NOTATION Décrit des données non XML. L'attribut attend une notation qui apparaît dans la DTD dans une déclaration NOTATION (voir ci-dessous).

```
<!ATTLIST image format NOTATION (TeX|TIFF)...>
```

ENTITY La valeur de l'attribut est une entité externe non XML (voir ci-dessous).

ENTITIES La valeur de l'attribut est une liste d'entités externes non XML.

```
<!ATTLIST album photos ENTITIES ... >
photos="pic1 pic2"
```

9 Déclaration d'entités

Les entités générales (sauf les entités prédéfinies) sont définies dans la DTD et utilisées dans les documents correspondants. Les entités paramètres sont définies dans la DTD et utilisées dans la DTD elle-même.

9.1 Les entités générales

Il s'agit de définir des raccourcis qui seront utilisés dans les documents XML liés à la DTD. Certaines entités sont déjà définies en XML comme : < (<), > (>), & (&), " ("), et ' (').

Déclaration d'une entité interne :

```
<!ENTITY nom_entité "texte de remplacement">
```

"texte de remplacement" ne doit pas contenir de ". Le texte de remplacement peut contenir des balises.

```
<!ENTITY ADN "acide désoxyribonucléique">
```

Pour exploiter cette entité dans le document XML, on écrit

```
<elt> L'\&ADN; est une molécule complexe ... </elt>
```

Déclaration d'une entité externe :

```
<!ENTITY nom_entité SYSTEM "uri_remplacement">
```

Dans la DTD :

```
<!ENTITY texte SYSTEM "fichier.txt">
```

Dans chaque document : &texte ; sera remplacé par le texte contenu du fichier `fichier.txt`.

```
<!ENTITY piedPage
   "<hr /><p>Copyright 2020, dernière mise à jour avril 2014.</p>">
```

Si le code associé à une entité devient très important, il peut être intéressant de le détacher dans un fichier à part, ce que permet la syntaxe suivante :

```
<!ENTITY piedpage SYSTEM "pp.xml">
```

Les entités permettent aussi d'associer un alias à un caractère non disponible sur le clavier.

```
<!ENTITY euro '&#x20AC;'>
```

Dans un document XML on écrit `&euro;`.

9.2 Les entités paramètres

Les entités paramètres doivent êtres définies dans une DTD et être utilisées dans la même DTD, soit pour éviter des répétitions dans les définitions des éléments ou des attributs, soit pour intégrer des fragments de DTD mémorisés dans un fichier, ce qui est utile pour fractionner les grosses DTD. Une entité paramètre doit obligatoirement être définie avant d'être utilisée. Le symbole % est utilisé dans la définition d'une entité paramètre pour la distinguer de la définition d'une entité générale.

```
<!ENTITY % nom_entité "texte de remplacement">
```

Exemple

```
<!ENTITY % commun "nom, prenom">
<!ELEMENT etudiant (%commun;, filiere+)>
<!ELEMENT professeur (%commun;, grade+)>
<!ELEMENT administrateur (%commun;, fonction)>
```

est équivalent à

```
<!ELEMENT etudiant (nom, prenom, filiere+)>
<!ELEMENT professeur (nom, prenom, grade+)>
<!ELEMENT administrateur (nom, prenom, fonction)>
```

Déclaration d'une entité externe :

```
<!ENTITY % nom_entité SYSTEM "uri_remplacement">
```

pour importer deux parties de DTD sauvegardées dans les fichiers partie1.dtd et partie2.dtd.

```
<!ELEMENT elt ( ... )>
<!ENTITY % partieDebut SYSTEM "partie1.dtd">
%partieDebut;
<!ENTITY % partieSuite SYSTEM "partie2.dtd">
%partieSuite;
```

Par exemple :

```
<!ENTITY Inclusion SYSTEM "fichier.xml">
<!ENTITY % Inclusion SYSTEM "fichier.inc">
```

Dans le fichier XML, le contenu du fichier fichier.xml sera inséré à l'appel de l'entité &Inclusion; et dans la DTD, le contenu du fichier fichier.inc sera inséré à l'appel de l'entité &Inclusion;

10 Déclaration de notations

Les notations permettent d'identifier par un nom le format des entités non-analysées par le parseur XML, les fichiers binaires par exemple (images, vidéos). Les notations permettent de définir le format des données et les applications qui permettent de les traiter.

```
<!NOTATION nomNotation SYSTEM|PUBLIC "notation">
```

XML permet de définir une application par défaut à lancer pour ouvrir des documents non XML; binaires par exemple. Ainsi, il est possible d'associer les images JPG au programme Paint Shop Pro (psp.exe) grâce à la syntaxe suivante :

```
<!NOTATION jpg SYSTEM "psp.exe">
```

Exemple : déclaration de notation associant le format nommé "flash" l'application spécifiée

```
<!NOTATION flash SYSTEM "/usr/bin/flash.exe">
```

Un nom de notation ne doit être défini qu'une seule fois dans un document.

Exemples :

```
<!NOTATION gif PUBLIC "image gif" "gif viewer">
<!NOTATION jpg SYSTEM "image.jpg">
```

Utilisation

```
<!ATTLIST balise attribut NOTATION (gif | jpg) #IMPLIED>
<!ENTITY image SYSTEM "image.gif" NDATA gif>
```

La valeur d'un attribut de type NOTATION est le nom d'une des notations déclarées dans la DTD et énumérées pour l'attribut.

11 INCLUDE et IGNORE

Les instructions (INCLUDE et IGNORE) permettent de prendre en compte un bloc de la DTD, ou de l'ignorer. La syntaxe est la suivante :

La syntaxe est la suivante :

```
<![IGNORE[
... déclaration à ignorer ...
]] >
```

```
<![INCLUDE[
... déclaration à inclure ...
]] >
```

Exemple :

```
<![IGNORE[
<!ELEMENT personne (nom,prénom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
]]>
```

Ce mécanisme devient puissant avec l'utilisation conjointe d'entités-paramètres.

```
<!ENTITY % bloc1 "INCLUDE">
<!ENTITY % bloc2 "IGNORE">
<![%bloc1;[
<!ELEMENT personne (nom,prénom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
]]>
<![%bloc2[
<!ELEMENT entreprise (dénomination,taille)>
<!ELEMENT dénomination (#PCDATA)>
<!ELEMENT taille (#PCDATA)>
]]>
```

12 Outils de validation

12.1 Validateur XML

Un analyseur syntaxique, parser, parseur ou processeur XML, est un programme informatique développé en Java, Perl, C++, Python ... qui permet :

– de vérifier qu'il est bien formé

– éventuellement vérifier sa validité par rapport à un schéma (DTD, W3C Schema, RelaxNG).

La validation permet de vérifier qu'un document XML est bien conforme à une DTD, (Schema XML ou autres). La conformité concerne un certain nombre de règles comme la structure des éléments XML, la présence des attributs, les cardinalités, les types de données etc... On dispose de nombreux validateurs, en ligne de commande ou en ligne et certains éditeurs permettent aussi la validation, comme NotePad++, Visual Studio et autres.

Un document XML bien formé est dit valide pour un schéma donné s'il respecte toutes les déclarations imposées par ce schéma. Les parseurs validant comparent les documents XML à leurs schémas et indiquent où le document diffère des contraintes spécifiées dans la DTD. Le caractère bien formé est requis pour tout document XML, la validité ne l'est pas toujours.

Commande `xmllint` sous Linux et Windows à saisir dans un terminal. La syntaxe est
xmllint [options] fichier1 fichier2 ...

`xmllint fichier.xml` : (sans aucune option) pour s'assurer que le fichier XML est bien formé.

Pour valider un fichier XML ayant une DTD interne :

```
xmllint --valid --noout fichierXML_DTD.xml
```

Pour valider un fichier XML ayant une DTD externe :

```
xmllint --dtdvalid fichier.dtd --noout fichierSansDTD.xml
```

L'option `--noout` empêche `xmllint` d'afficher le fichier XML.

Pour plus de détails sur `xmllint` et ses options utiliser la commande `man xmllint` sous Linux.

Il y a plusieurs validateurs en ligne. Par exemples :

`https://validator.w3.org/`

`https://www.xmlvalidation.com/`

Université de Brown : `http://my2lbox.com/fr/validateur-xml` .

`http://www.stg.brown.edu/service/xmlvalid/`

13 Conclusion

Limites de DTD :

- On ne peut contrôler le nombre d'éléments contenu dans une balise.
- absence de type de contenu d'un élément (nombre, texte, date etc) ;
- On ne peut contrôler les types (entier, décimal ...) de valeurs des attributs.
- On ne peut décrire ses propres types de données dans une DTD.
- On ne peut faire des références vers d'autres DTD.
- ne spécifie pas l'élément racine, un document peut être valide en utilisant n'importe quelle balise de la DTD comme racine ;
- le nombre d'apparitions d'un élément ne peut être contraint précisément ;
- on ne peut pas contraindre la forme de ces contenus (entre 5 et 20 caractères, contenant un signe , etc.) ;
- enfin, le langage utilisé pour définir une DTD n'est pas un langage XML !
- les DTD ne supportent pas les "espaces de nom" .

Pour palier à ces manquements, d'autres propositions ont été faites, permettant de spécifier un langage XML, citons les XML Schema et Relax NG.

Le langage XML Schema Definition (XSD) ou XML Schema a été développé par W3C [1] en 2001, est un langage de définition de modèles de documents XML. Un modèle est la grammaire ou la description de la structure que doit respecter un ou plusieurs documents XML. Une telle grammaire ou description est décrite en langage XML Schema et s'appelle schéma XML. Son vocabulaire est composé d'environ 30 éléments et attributs et 44 types de données prédéfinis.

XSD est actuellement la norme de facto pour décrire les documents XML. Il est extrêmement complet et, par conséquent, relativement complexe.

Un schéma XML défini pour un document XML :
- les éléments ainsi que leurs attributs ;
- les types des éléments et des attributs ;
- la hiérarchie entre les éléments ;
- l'ordre des éléments ;
- l'occurrence précise de chaque élément ;
- les valeurs par défaut des éléments et des attributs ;
- le format ou la restriction des valeurs d'un élément ou attribut.

Le langage XML Schema est destiné à remplacer le langage DTD. Il est beaucoup plus évolué et précis que le langage DTD et s'écrit en plus avec la syntaxe XML et exploite les espaces de noms.

Un document XML est valide pour un schéma XML, si il est bien formé, tous les éléments et attributs de ce document sont déclarés dans le schéma et si il respecte toutes les contraintes imposées par ce schéma.

1. http ://www.w3.org/XML/Schema

14 Structure d'un schéma XML

Un Schéma XML est un fichier XML bien formé, doit donc commencer par un prologue XML et de l'élément racine imposé **<xsd:schema>** et dont le contenu est une suite de définitions d'éléments, d'attributs et de leurs types.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
      <!-- ici déclarations d'éléments, d'attributs de types -->
</xsd:schema>
```

Un Schéma XML est un fichier texte dont l'extension est ".xsd". La liaison d'un schéma XML avec un fichier XML se fait grâce à deux attributs de l'élément racine du fichier XML. L'espace de noms : **xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"** et le second attribut permet d'indiquer au fichier XML le nom et où se trouve le fichier contenant le Schéma XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<racine xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="chemin_vers_fichier.xsd">
... contenu du fichier XML ...
</racine>
```

15 Les types prédéfinis en XML Schema

La notion de type est fondamentale en XML Schema. Il y en a 44 types prédéfinis, appelés aussi types de base. Et on a la possibilités de définir de nouveaux types.

Les types prédéfinis sont regroupés en 4 groupes : types chaînes de caractères, types numériques, types dates et autres types.

15.1 Les types chaînes de caractères

Certains types pour les attributs sont hérités des DTD. Ces types facilitent la traduction automatique des DTD en XML Schema. Pour des raisons de compatibilité, ces types sont réservés aux attributs. Voir Table 2.1.

15.2 Les types numériques

Sont listés dans la Table 2.2.

15.3 Les types dates

Sont listés dans la Table 2.3.

Type	Description
xsd:string	représente une chaîne de caractères unicode.
xsd:normalizedString	représente chaîne de caractères ne contenant pas de tabulation, de saut de ligne ou de retour chariot.
xsd:token	chaîne de caractères de type **xsd:normalizedString** et ne contenant pas en plus des espaces en début ou en fin ou des espaces consécutifs.
xsd:language	représente le code d'une langue naturelle d'après le code ISO 639 par exemple : en English, en-GB UK English, en-US US English, fr French, de German, es Spanish
xsd:NMTOKEN	représente une chaîne de caractères unique ne contenant pas d'espace, peut contenir lettres, chiffres, point, tiret, tiret bas, deux points.
xsd:NMTOKENS	représente une liste de valeurs NMTOKEN séparées par espace. La liste doit contenir au moins un **xsd:NMTOKEN**.
xsd:Name	représente un nom XML qui peut être utilisé comme nom de type d'élément ou nom d'attribut
xsd:NCName	représente un nom XML ne contenant pas le caractère :
xsd:ID	utilisé pour un attribut qui identifie un élément de façon unique dans un document XML. La valeur d'un **xsd:ID** doit être un NCName.
xsd:IDREF	utilisé pour un attribut qui fait référence à un ID. Tous les attributs de type xsd :IDREF doit faire référence à un ID dans le même document XML.
xsd:IDREFS	représente une liste de valeurs IDREF séparées par espace. La liste doit contenir au moins un IDREF. Chaque IDREF doit faire référence à un ID dans le même document XML
xsd:ENTITY	représente une référence à une entité non parsée. La valeur d'un xsd :ENTITY doit être un NCName. xsd :ENTITY est souvent utilisé pour inclure des données non XML, comme des graphiques.
xsd:ENTITIES	représente une liste d'entités représente une liste de valeurs ENTITY séparées par espace. La liste doit contenir au moins un ENTITY. Chaque valeur de ENTITY doit correspondre à un nom de ENTITY non parsée déclaré.

TABLE 2.1 – Les types chaînes de caractères en XML Schema

Type	Description
xsd:byte	Nombre entier signé sur 8 bits
xsd:unsignedByte	Nombre entier non signé sur 8 bits
xsd:short	Nombre Nombre entier signé sur 16 bits
xsd:unsignedShort	Nombre entier entier non signé sur 16 bits
xsd:int	Nombre entier signé sur 32 bits
xsd:unsignedInt	Nombre entier non signé sur 32 bits
xsd:long	Nombre entier signé sur 64 bits. Ce type dérive du type **xsd:integer**
xsd:unsignedLong	Nombre entier non signé sur 64 bits
xsd:integer	Nombre entier sans limite de précision Ce type n'est pas primitif et dérive du type **xsd:decimal**
xsd:positiveInteger	Nombre entier strictement positif sans limite de précision
xsd:negativeInteger	Nombre entier strictement négatif sans limite de précision
xsd:nonPositiveInteger	Nombre entier négatif ou nul sans limite de précision
xsd:nonNegativeInteger	Nombre entier positif ou nul sans limite de précision
xsd:float	Nombre flottant sur 32 bits conforme à la norme IEEE 754
xsd:double	Nombre flottant sur 64 bits conforme à la norme IEEE 754
xsd:decimal	Nombre décimal sans limite de précision

TABLE 2.2 – Les types numériques en XML Schema

Type	Description
xsd:duration	représente une durée sous la forme PyYmMdDThHm'MsS où P commence le format, y le nombre d'années suivi de Y, m le nombre de mois suivi de M, d le nombre de jours suivi de D, T sépare la date et le temps, h le nombre d'heures suivi de H, m' le nombre de minutes suivi de M, et s le nombre de secondes suivi de S. Toute valeur nulle peut être omise ainsi la lettre qui la suit. T ne doit pas apparaitre s'il n'y a pas d'heure, minutes et secondes. Une durée peut être négative en précédant P par le symbole moins. Par exemple PT10M représente une durée de 10mn. -P2D représente une durée négative de 2 jours.
xsd:date	représente une date sous la forme yyyy-mm-dd. Par exemple 2018-04-10 est le 10 avril 2018. -0175-01-01 est le premier janvier de l'année, 175 av. J-C.
xsd:time	représente l'heure sous la forme hh :mm :ss.sss. Par exemple 00 :00 :00 minuit.
xsd:dateTime	représente une date et une heure pécifiques au format YYYY-MM-DDThh :mm :ss.sss, Par exemple : 2004-04-01T08 :20 :00
xsd:gYear	représente une année. Par exemple 2018.
xsd:gYearMonth	représente un mois spécifique d'une année spécifique. Le format de **xsd:gYearMonth** est YYYY-MM.
xsd:gMonth	représente un mois remarquable de l'année. Le format de **xsd:gMonth** est –MM. Par exemple –03 est le mois de mars.
xsd:gMonthDay	représente un jour de l'année remarquable et qui se répète. Comme les anniversaires. Le format de **xsd:gMonthDay** est –MM-DD. Par exemple : –04-01 représente le premier avril 12.
xsd:gDay	représente un jour

TABLE 2.3 – Les types Date en XML Schema

Type	Description
xsd:boolean	permet de n'accepter que true, false, 1 et 0 comme valeurs dans le document. TRUE et False ne sont pas acceptés.
xsd:QName	représente un espace de noms XML qualifiés. pre :monElement
xsd:NOTATION	représente une référence à une notation Une notation est une méthode d'interprétation de contenu XML et non-XML. Pour exemple, si un document XML contientune image JPEG, une notation peut être déclarée pour indiquer que c'est une donnée de type JPEG. La valeur de **xsd:NOTATION** doit être un QName. C'est le seule type prédéfini qui ne peut être le type d'élément ou d'attribut.
xsd:anyURI	représente une URI
xsd:base64Binary	représente une donnée binaire au format Base64
xsd:hexBinary	représente une donnée binaire au format hexadécimal : sont autorisés a..z, A..Z, O..9, +, /, = et espace. Le nombre des caractères différents de espace doit être un multiple de 4 sinon on complète par = à la fin : un = doit être précédé par un des caractères AQgw ; deux = doit être précédé par un des caractères AEIMQUYcgkosw048.

TABLE 2.4 – Les autres types en XML Schema

15.4 Les autres types

Sont listés dans la Table 2.4.

Exemple d'utilisation d'un type simple prédéfini

```
<xsd:attribute name="taille" type="xsd:decimal"/>
<xsd:attribute name="age" type="xsd:positiveInteger"/>
<xsd:attribute name="date_naissance" type="xsd:date"/>
```

Dans la suite, il s'agirait de définir de nouveaux types à partir de types prédéfinis.

16 Déclaration d'éléments simples et d'attributs

Rappelons que tout élément et attribut devant apparaître dans le document XML doit être déclaré dans le schéma.

On distingue deux catégories d'éléments, simples et complexes :

Les éléments simples, se sont les éléments qui n'ont pas d'attribut et leur contenu est uniquement du texte. Les attributs sont aussi considérés simples.

Les éléments complexes sont :
les éléments vides ;
les éléments ayant des attributs et qui contiennent uniquement du texte ;
les éléments qui contiennent d'autres éléments ;
les éléments qui contiennent du texte et d'autres éléments.

16.1 Déclaration d'éléments simples

L'élément **<xsd:element>** permet de déclarer les éléments dans un schéma XML. Il doit être enfant direct de l'élément **<xsd:schema>**. Chaque élément déclaré est associé à un type de données par l'attribut **type**. Le type est soit un type prédéfini (comme **xsd:integer**, **xsd:string** ...), soit un type défini globalement dans le schéma, soit un type défini explicitement dans la déclaration de l'élément (que nous allons voir plus tard).

Syntaxe de déclaration d'élément simple

```
<xsd:element name="..." type="..."/>
```

L'élément **<xsd:element>** a plusieurs attributs dont :

name : indique le nom de l'élément XML.

type : indique le type de l'élément. Lorsqu'on n'attribue pas de type à un élément, il est considéré comme de type **xsd:anyType** et peut donc contenir n'importe quoi.

On peut préciser le nombre d'apparition autorisé d'un élément à l'aide des attributs **minOccurs** et **maxOccurs** ;

maxOccurs : l'attribut précise le nombre d'occurrences maximum de l'élément. Par défaut, ce nombre est égal à 1. **maxOccurs** prend un entier strictement positif ou la chaîne **unbounded** comme valeur pour indiquer qu'il n'y a pas de nombre maximal.

minOccurs : précise le nombre d'occurrences minimum de l'élément. Par défaut, ce nombre est égal à 1.

Puisque la valeur par défaut de **minOccurs** et **maxOccurs** est 1, si ces deux attributs sont absents alors l'élément doit apparaître une fois et une seule.

default : lorsque l'élément est simple et présent mais avec un contenu vide, il est possible de donner une valeur par défaut.

Exemple de déclaration d'un élément simple.

```
<xsd:element name="title"
             type="xsd:string"
             default="Titre à préciser plus tard"
             minOccurs="2"
             maxOccurs="5"/>
```

```
En XSD : <xsd:element name="naissance" type="xsd:dateTime"/>

En XML :
<naissance>
      2022-01-12T12:13:14
</naissance>
```

16.2 Déclaration d'attributs

L'élément **<xsd:attribute/>** permet de définir les attributs d'éléments. Un attribut est toujours considéré de type simple. Par défaut, un attribut est optionnel. Un attribut fournit des informations supplémentaires à un élément. Contrairement aux DTD, les attributs ne sont pas directement associés aux éléments. Ils font partie des types qui sont donnés aux éléments.

Un attribut est généralement spécifié dans la définition XSD d'un élément, ce qui le lie à cet attribut. Les attributs peuvent également être définis globalement puis rattacher à différents éléments.

Syntaxe de déclaration d'attributs

```
<xsd:attribute name="..." type="..."/>
```

name : obligatoire, indique le nom de l'attribut XML.

type : fournit le type de données accepté par l'attribut.

id : précise un identificateur unique pour l'attribut.

use : précise le caractère obligatoire ou facultatif de l'attribut. Valeurs possibles pour use :

required : l'attribut doit apparaître et prendre une valeur.

prohibited : l'attribut ne doit pas apparaître. La valeur prohibited est utile dans les restrictions de types pour modifier l'utilisation d'un attribut.

optional : l'attribut peut apparaître et prendre une valeur quelconque. Cette valeur est très peu utilisée car c'est la valeur par défaut.

default : si l'attribut à une valeur définie il la prend sinon il prend la valeur par défaut.

fixed : signifie que la valeur dans le document XML ne peut avoir que la valeur spécifiée dans le XSD par cet attribut.

On ne peut donc spécifier simultanément les deux attributs **default** et **fixed**.

Par exemple, on suposant que les types **fontSize** et **jourSemaine** sont définis (nous allons voir comment définir de nouveaux types)

```
<xsd:attribute name="size" type="fontSize" use="required"/>
<xsd:attribute name="jour" default="lundi" type="jourSemaine"/>
<xsd:attribute name="version" type="xsd:number" fixed="1.0"/>
<xsd:attribute name="rdv" type="xsd:date" use="required"
```

```
                                                    default="2003-01-01"/>
```

On peut regrouper un ensemble d'attributs et de l'identifier par un nom. Et cet ensemble peut être associé à n'importe quel élément du schéma. Une définition de groupe d'attributs doit se faire au niveau global et être référencés explicitement dans la définition d'éléments. Les attributs ne peuvent être rattaché qu'à des éléments complexes. Voir Section 17.1. Pour cela nous utilisons **<xsd:complexType>**.

Exemple :

```
<xsd:attributeGroup name="attributsImage">
      <xsd:attribute name="source" type="nomfichier"/>
      <xsd:attribute name="alt" type="xsd:string"/>
</xsd:attributeGroup>
```

16.3 Définition de nouveaux types simples

Les types simples définissent uniquement des contenus textuels. Ils peuvent être utilisés pour les éléments ou les attributs. Un attribut est toujours de type simple. Les types simples sont introduits par l'élément **<xsd:simpleType>**. Un type simple est souvent obtenu par restriction d'un autre type prédéfini. Il peut aussi être construit par union d'autres types simples ou par l'opérateur de listes.

L'élément **<xsd:simpleType>** permet de définir, à partir de types prédéfinis, de nouveau types simples pour les attributs et les éléments simples. Il peut avoir un attribut **name** si la déclaration est globale. Le type ainsi défini peut être réutilisé autant de fois qu'on le veut. La syntaxe est :

```
<xsd:simpleType name="...">
      ...
</xsd:simpleType>
```

La déclaration local d'un type à un élément consiste à attacher un type à un élément sans nommer ce type (on parle parfois de type anonyme). On utilise alors la syntaxe

```
<xsd:element name="monElement">
      <xsd:simpleType>
      ...
      </xsd:simpleType>
</xsd:element>
```

```
<xsd:attribute name="monAttribut">
      <xsd:simpleType>
      ...
      </xsd:simpleType>
</xsd:attribute>
```

Un type simple est souvent obtenu par restriction, par : Restriction, List or Union.

Une union est un mécanisme permettant de combiner 2 ou plusieurs types de données différents en un seul.

Une liste permet à la valeur (dans le document XML) de contenir un certain nombre de valeurs valides séparées par des espaces.

Dérivation par restriction de domaine

La dérivation par restriction consiste à créer un type dont l'ensemble de valeurs est inclus dans l'ensemble de valeurs d'un type existant.

La restriction de l'espace des valeurs d'un type pré-existant, est définie par des contraintes de facettes du type de base : valeur min, valeur max ... énumération d'un ensemble de valeurs ou par expression régulière

On utilise l'élément **<xsd:restriction>** dont l'attribut **base** indique l'ensemble à restreindre. La syntaxe est :

Exemple :

```
<xsd:simpleType name="ChiffresOctaux">
      <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="0"/>
            <xsd:maxInclusive value="7"/>
      </xsd:restriction>
</xsd:simpleType>
```

Pour définir un type qui prend ses valeurs dans un ensemble précis, on utilise l'élément **<enumeration/>** dont l'attribut **value** décrit les valeurs une par une. Par exemple :

```
<xsd:simpleType name="jourSemaine">
   <xsd:restriction base="xsd:token">
     <xsd:enumeration value="lundi"/>
            ...
     <xsd:enumeration value="dimanche"/>
   </xsd:restriction>
</xsd:simpleType>
```

On peut aussi utiliser les expressions régulières pour définir des formats de chaîne de caractères. Pour cela on utilise **<xsd:pattern/>** dont l'attribut **value** permet de décrire le format.

```
<xsd:simpleType name="numTel">
      <xsd:restriction base="xsd:string">
            <xsd:pattern value="+212-\(0\)\d(-\d{2}){4}"/>
      </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name='non-empty-string'>
      <xsd:restriction base='string'>
            <xsd:minLength value='1'/>
      </xsd:restriction>
</xsd:simpleType>

<xsd:length value="30">
```

Dérivation par liste

Comment contraindre un élément à contenir des données sous forme de liste (séparées par des espaces) ?

par exemple **<debits> 4 8 12 16 </debits>**

L'élément **<list>** permet de définir un nouveau type dont une valeur est une liste d'éléments d'un type pré-existant indiqué par l'attribut **itemType**.

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="debits" type="DebitsPossibles"/>
      <xsd:simpleType name="DebitsPossibles">
            <xsd:list itemType="xsd:nonNegativeInteger"/>
      </xsd:simpleType>
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<debits>
      4 8 12 16
</debits>
```

Exemples de types list

Dérivation par union

L'élément **<xsd:union>** permet de faire l'union ensembliste de toutes les valeurs possibles de plusieurs types existants ou unir deux types en cours de leurs définition.

Très utile pour valider une donnée qui pourrait être de plusieurs types possibles.

Dans le cas de types déjà définis, l'attribut **memberType** de l'élément **<xsd:union/>** prend comme valeur les différents noms de types à unir séparés par espace.

Exemple de type union

```
<!-- permet de faire <font size="12"> ou <font size="medium"> -->
<xsd:simpleType name="fontSize">
      <xsd:union>
```

```
            <xsd:simpleType>
                  <xsd:restriction base="xsd:positiveInteger">
                        <xsd:minInclusive value="8"/>
                        <xsd:maxInclusive value="72"/>
                  </xsd:restriction>
            </xsd:simpleType>

            <xsd:simpleType>
                  <xsd:restriction base="xsd:NMTOKEN">
                        <xsd:enumeration value="small"/>
                        <xsd:enumeration value="medium"/>
                        <xsd:enumeration value="large"/>
                  </xsd:restriction>
            </xsd:simpleType>
      </xsd:union>
</xsd:simpleType>
```

Exemple d'union de types déjà définis. On utilise l'attribut **memberTypes** de **<union>**. Les types à unir sont séparés par des espaces.

```
<xsd:simpleType name="TransportFormatCaracteres">
      <xsd:union memberTypes="base64Binary hexBinary"/>
</xsd:simpleType>
```

17 Définition d'éléments complexes

Rappelons que, sont de type simples les attributs et les les éléments simples c'est à dire les éléments sans attributs et qui ont un contenu mais ne contenant pas de sous éléments. Tout élément qui n'est pas simple est alors qualifié d'élément complexe. Ainsi le type complexe concerne uniquement les éléments.

La construction explicite d'un élément de type complexe se fait grâce à l'élément **<xsd:complexType>**. Il ne peut être inclus que dans les éléments suivants : **<schema>**, **<element>** ou **<redefine>**.

<xsd:complexType> déclare tout élément qui peut avoir des attributs et/ou des sous-éléments simples en plus de trois sortes d'enfants à réaliser par les opérateurs suivants :

– opérateur de séquence **<xsd:sequence>**, pour déclarer des sous-éléments qui doivent être dans cet ordre dans les documents XML.

– opérateur de choix **<xsd:choice>**, pour déclarer des sous-éléments et le document XML doit contenir un seul de ces éléments.

– opérateur d'ensemble **<xsd:all>**, le document à valider doit contenir certains de ces éléments et dans l'ordre qu'on veut.

Les attributs de l'élément **<complexType>** sont :

name : indique le nom de l'élément XML.

mixed : pour un contenu mixte entre sous éléments et texte. Les valeurs de cet attribut sont **"true"** et **"false"** qui est la valeur par défaut.

default : précise une valeur par défaut pour l'élément.

id : précise un identificateur unique pour l'élément.

17.1 Attacher des attributs à un élément

Si on a déjà défini des attributs individuellement par **<attribute>** ou un groupe d'attributs par **<attributeGroup>** on peut les attacher à un ou plusieurs éléments en utilisant les éléments **<xsd:complexType>**, **<attribute>** et son attribut **ref**. La valeur de **ref** doit être un QName et la référence peut inclure un préfixe d'espace de noms. Comme dans l'exemple suivant :

```
<xsd:attribute name="global" type="xs:string"/>
<xsd:attributeGroup name="attributsImage">
      <xsd:attribute name="source" type="nomfichier"/>
      <xsd:attribute name="alt" type="xsd:string"/>
</xsd:attributeGroup>
```

```
<xsd:element name="image">
      <xsd:complexType>
            <xs:attribute ref="global"/>
            <xsd:attributeGroup ref="attributsImage"/>
            <xsd:attribute name="titre" type="xsd:string"/>
      </xsd:complexType>
</xsd:element>
```

17.2 Sous-éléments ordonnés

On utilise l'élément **<xsd:sequence>**. Les éléments de la séquence doivent apparaître dans l'ordre où ils sont déclarés. C'est l'équivalent de ',' en DTD.

Les valeurs par défaut pour **minOccurs** et **maxOccurs** est 1. Ainsi, si les attributs **minOccurs** et **maxOccurs** sont absents alors l'élément doit apparaître une et une seule fois.

De plus le nombre d'occurrences de chaque sous-élément peut être défini par les attributs **minOccurs** et **maxOccurs**.

```
<xsd:complexType name="typeDuree">
      <xsd:sequence>
            <xsd:element name="du" type="xsd:date"/>
            <xsd:element name="au" type="xsd:date"/>
      </xsd:sequence>
```

```
</xsd:complexType>

<xsd:element name="duree" type="typeDuree"/>
```

En XML on doit écrire :

```
<duree>
      <du>2010-09-13</du>
      <au>2010-09-25</au>
</duree>
```

Exemple :

```
<xsd:attribute name="src" type="xsd:anyURI"/>

<xsd:element name="image">
      <xsd:complexType>
            <xsd:attribute ref="src"/>
      </xsd:complexType>
</xsd:element>

<xsd:element name="images">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="image" maxOccurs="2"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

17.3 Sous-éléments au choix

On utilise à cet effet l'élément **<xsd:choice>**. Le document à valider doit contenir seulement un des éléments du choix (l'équivalent de '|' en DTD). Le nombre d'occurrences possibles pour chaque sous-élément est déterminé par les attributs **minOccurs** et **maxOccurs** de l'élément.

```
<xsd:element name="pieceIdentite">
      <xsd:complexType>
            <xsd:choice>
                  <xsd:element name="cin" type="xsd:string"/>
                  <xsd:element name="passeport" type="xsd:string"/>
            </xsd:choice>
      </xsd:complexType>
</xsd:element>
```

On peut imbriquer **<xsd:choice>** dans **<xsd:sequence>**. Par exemple :

```
<xsd:complexType name="ElemPersonne">
      <xsd:sequence>
            <xsd:element name="prénom" type="xsd:string"/>
            <xsd:element name="nom" type="xsd:string"/>
            <xsd:choice>
                  <xsd:element name="age" type="xsd:string"/>
                  <xsd:element name="date_naiss" type="xsd:date"/>
            </xsd:choice>
      </xsd:sequence>
</xsd:complexType>
```

17.4 Sous-éléments non-ordonnés

L'élément **<xsd:all>** n'a pas d'équivalent simple dans les DTD. Il est utilisé pour décrire un groupe non ordonné d'éléments dont le nombre d'occurrences peut être zéro ou un.

Il doit être un enfant direct de l'élément **<xsd:complexType>** ou **<xsd:complexContent>**.

L'opérateur **<xsd:all>** ne peut pas être imbriqué avec **<xsd:sequence>**, **<xsd:choice>** ou même **<xsd:all>**. Le seul enfant possible de **<xsd:all>** est **<xsd:element>**.

Les attributs **minOccurs** et **maxOccurs** de **<xsd:all>** ne peuvent pas avoir des valeurs quelconques. La valeur de l'attribut **minOccurs** doit être 0 ou 1 et la valeur de l'attribut **maxOccurs** doit être 1 qui est la valeur par défaut.

```
<xsd:element name="personne">
      <xsd:complexType>
            <xsd:all>
                  <xsd:element name="nom" type="xsd:string"/>
                  <xsd:element name="prenom" type="xsd:string"/>
                  <xsd:element name="surnom" type="xsd:date"/>
            </xsd:all>
      </xsd:complexType>
</xsd:element>
```

17.5 Définition par extension

L'extension consiste à ajouter à un type existant (simple ou complexe) des sous-éléments ou des attributs. On obtient alors bien évidemment un type complexe.

L'extension est réalisée par l'élément **<xsd:extension>** qui a pour attribut obligatoire **base** pour indiquer le nom du type (prédéfini, simple ou complexe) à étendre. Cet élément est un sous élément de **<xsd:simpleContent>** ou de **<xsd:complexContent>**.

`<xsd:simpleContent>` défini un type simple et spécifie les contraintes et les informations sur les valeurs d'attributs ou le contenu uniquement textuel des éléments.

`<xsd:complexContent>` défini extension ou restriction sur un type complexe à contenu mixe (texte et sous éléments) ou à contenu uniquement des sous éléments.

Types simples

L'extension d'un simple permet d'ajouter des attributs pour donner un type complexe à contenu simple.

Le code de schéma suivant définit un type `Price` qui étend le type prédéfini `xsd:decimal` en lui ajoutant un attribut currency de type `xsd:string`

```
<xsd:complexType name="Price">
<xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
            <!-- Attribut ajouté -->
            <xsd:attribute name="currency" type="xsd:string"/>
      </xsd:extension>
</xsd:simpleContent>
</xsd:complexType>

<xsd:element name="price" type="Price"/>
```

L'expression

```
<price currency="euro">3.14</price>
```

est valide.

Types complexes à contenu simple. Il est possible d'étendre un type complexe à contenu simple pour lui ajouter de nouveaux attributs. On obtient alors un nouveau type complexe à contenu simple qui possède les attributs du type de base et ceux déclarés par l'extension.

Types complexes à contenu complexe. L'extension d'un type complexe à contenu complexe consiste à ajouter du contenu et/ou des attributs. Le contenu est ajouté après le contenu du type de base.

```
<xs:complexType name="address">
      <xs:sequence>
            <xs:element name="street" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
      </xs:sequence>
</xs:complexType>

<xs:complexType name="USAddress">
      <xs:complexContent>
            <xs:extension base="address">
            <xs:sequence>
```

```
                    <xs:element name="state" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="country" type="xs:string" fixed="US"/>
            </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

17.6 Éléments à contenu mixte

Le contenu d'un élément est mixte s'il contient du texte (autre que des caractères d'espacement) en dehors de ses enfants. Pour créer un élément à contenu mixte on utilise **<complexType>** avec l'attribut **mixed="true"**.

```
<xsd:element name="personne">
      <xsd:complexType mixed="true">
            <xsd:sequence>
                  <xsd:element name="prenom" type="xsd:string"/>
                  <xsd:element name="nom" type="xsd:string"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

Le code suivant est valide

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<personne>
      Prénom :<prenom>Albert</prenom>,
      Nom :<nom>Einstein</nom>.
</personne>
```

17.7 Élément vide

Pour déclarer un élément vide on utilise **<xsd:complexType>** pour déclarer uniquement les attributs de cet élément. L'élément **
** en XHTML contient uniquement des attributs. Les types utilisés sont définis dans le schéma de XHTML.

```
<xsd:element name="br">
<xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="class" type="xsd:NMTOKENS"/>
      <xsd:attribute name="style" type="StyleSheet"/>
      <xsd:attribute name="title" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
```

```
<xsd:element name="vide">
      <xsd:complexType/>
</xsd:element>
```

```
<xsd:element name="br">
      <xsd:complexType>
            <xsd:attribute name="class" type="string"/>
      </xsd:complexType>
</xsd:element>
```

18 Autres éléments

<xsd:annotation>

L'élément **<xsd:annotation>** permet d'ajouter des commentaires dans un schéma. Il peut être enfant de l'élément **<xsd:schema>** pour des commentaires globaux ou enfant des éléments **<xsd:element>**, **<xsd:attribute>** pour ajouter des commentaires à la déclaration d'éléments et d'atttributs ainsi que de **<xsd:simpleType>** et **<xsd:complexType>** pour ajouter des commentaires aux définitions de type.

L'élément **<xsd:annotation>** admet deux sous-éléments : **<xsd:documentation>** dont le contenu est à lire par un humain et **<xsd:appinfo>** dont le contenu est destiné aux applications. Ces deux éléments admettent un contenu mixte, c'est à dire un mélange de texte et d'éléments

<xsd:group>

L'élément **<xsd:group>** permet de définir un groupe d'éléments et d'y faire référence dans un schéma XML. Par exemple :

```
<xsd:element name="element1" type="xsd:string"/>
<xsd:element name="element2" type="xsd:string"/>

<xsd:attribute name="monAttribut" type="xsd:decimal"/>

<xsd:group name="lesElements">
      <xsd:sequence>
            <xsd:element ref="element1"/>
            <xsd:element ref="element2"/>
      </xsd:sequence>
</xsd:group>

<xsd:complexType name="lesElementsComplet">
```

```
      <xsd:group ref="lesElements"/>
      <xsd:attribute ref="monAttribut"/>
</xsd:complexType>
```

19 Les outils de validation

Plusieurs outils sont disponibles pour valider des documents XML par rapport à leurs Schémas XML. Certains sont à installer, d'autres en ligne, et il y a des éditeur XML qui font la validation aussi.

– La validation avec xmllint (à installer) la commande est :

xmllint –schema monSchema.xsd monDoc.xml

Validation en ligne : `http://xmlvalidation.com`

– des éditeurs comme `Oxygen` et `Visual Studio`

– Toutes plate-formes (java) : `http://xerces.apache.org/xerces2-j/`

– Autres outils : `http://www.w3.org/XML/Schema#Tools`

– Transformation DTD XML Schema :

`http://www.syntext.com/products/index.html#Dtd2Xs`

On valide le document par :

xmllint –noout –schema fichier.xsd document.xml xmlstarlet val –xsd schema.xsd -e document.xml