UE Ingénierie documentaire
Master 2ème Année

Fabrice Lefèvre

2021

AVIGNON UNIVERSITÉ

XML

Partie 2 - Syntaxe XML

Fabrice Lefèvre
fabrice.lefevre@univ-avignon.fr
2021

Structure d'un document XML

Un document XML comprend trois parties :

- le prologue, avec la déclaration XML, la DTD, des commentaires, des instructions de traitements (optionnelles)
- un élément racine avec son contenu
- un épilogue avec des commentaires, ou des instructions de traitements (optionnelles)

- Le *contenu du document* proprement dit est le contenu de l'élément racine

Déclaration XML

Tout document XML peut être précédé par une déclaration définissant le langage utilisé :

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

- l'attribut encoding indique le jeu de caractères utilisé dans le document
- l'attribut optionnel standalone indique si le document est composé de plusieurs entités.

Déclaration de type

On peut indiquer qu'un document est conforme à une *DTD* :

```
<!DOCTYPE nom SYSTEM "sourceExt" [decLoc]>
```

- nom est le type de l'élément racine
- sourceExt est un source extérieure contenant la DTD
- decLoc sont des déclarations locales (pour les entités principalement)

Bien formé et valide

- **Bien formé** : respecte la syntaxe du XML

  Principalement :

  - `unique élément racine décomposable en éléments fils eux-mêmes décomposables en éléments`
  - `pas de chevauchement de balises`
  - `fermetures des balises`
- **Valide** : respecte les contraintes données par la grammaire définies dans la DTD ou le schéma XML

Eléments

Les **éléments** forment la structure même du document :

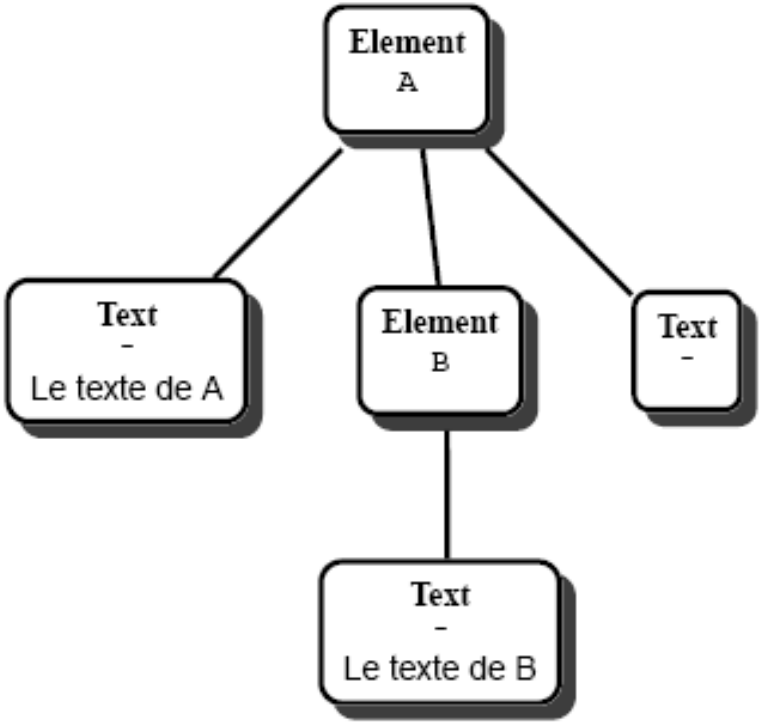
- Ce sont les branches et les feuilles de l'arborescence.
- Ils peuvent contenir du texte ou d'autres éléments, alors appelés "éléments enfants"
  - `l'élément contenant étant appelé "élément parent"`
- Dans la forme sérialisée :
  - `c'est une balise ouvrante avec un nom puis un contenu puis une balise fermante`
- Dans la forme arborescente :
  - `c'est un nœud avec un nom`
  - `le contenu est un arbre`

Eléments

Quelques remarques sur les éléments :

- tout document comprend un et un seul élément racine
- un nom d'élément ne contient pas de blancs, ni de caractères accentués
- les majuscules sont distinguées des minuscules
- il existe une forme abrégée pour les éléments sans contenu : <TAG></TAG> peut s'écrire <TAG />

Exemple : un élément avec contenu




```
<?xml version="1.0"
   encoding="ISO-8859-1"?>
<A>
Le texte de A
<B>
   Le texte de B
</B>
</A>
```

Attributs

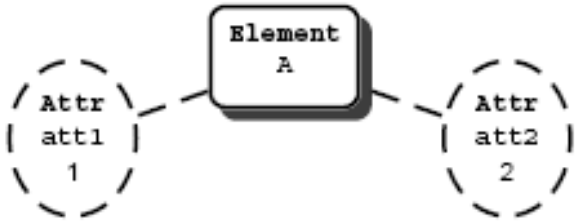
Les **attributs** des éléments constituent un autre moyen de représenter de l'information en l'associant directement à un élément :

```
<A att1='1' att2='2'> … </A>
```

Quelques remarques sur les attributs :

- l'ordre des attributs n'est pas pertinent
- il doit toujours y avoir une valeur, encadrée par des guillemets (contrairement à HTML)
- il ne doit pas y avoir deux attributs avec le même nom dans un élément
- il ne peuvent être déclarés que dans la balise ouvrante

Exemple d'attributs



- `<A att1='1' att2='2'>` et `<A att2='2' att1='1'>` sont équivalents
- `<A att=a>` : incorrect (manque apostrophes)
- `<A att1='1' att1='2'/>` : interdit
- `<A att1='1'/> <B att1='2'/>` : correct (même nom mais deux éléments différents)

Commentaires et instructions de traitement

- Les **commentaires** :
  - ne pas en abuser, sur le web il ne faut jamais alourdir les fichiers inutilement

```
<!-- Ceci est un commentaire -->
```

- Les **instructions de traitement** : une instruction interprétée par l'application servant à traiter le document document XML.
  - les instructions de traitement qui servent le plus souvent sont la déclaration XML ainsi que la déclaration de feuille de style :

```
<?xml-stylesheet href="prog.xslt" type="text/xslt">
```

Entités

Intuitivement, les entités peuvent être vues comme des raccourcis (ou des alias, macros) qui seront utilisables dans les documents XML liés à la DTD.

- La déclaration des entités s'effectue au sein de la DTD. Elles peuvent être utilisées aussi bien dans la DTD que dans le document XML.
- Plusieurs sortes d'entités :
  - définissables et définies
  - analysables ou non
  - internes ou externes.
- Exemple : entités caractères.
  - Certains caractères 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. On a recours dans ce cas à des entités prédéfinies.
    - "<" → <

Entités et références à des entités

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE A SYSTEM "minimal.dtd" [
<!ENTITY monTexte "texte simple">
<!ENTITY maSignature SYSTEM "signature.xml">
]>
<A>
Du &monTexte;, sans caractères réservés:
ni &lt; ni &gt; ni &amp; ni &apos; ni &quot;
&maSignature;
</A>
```

Les *entités* servent à factoriser des parties du document.

Sections littérales

À priori, on n'a pas le droit de placer dans le contenu d'un document XML des caractères spéciaux comme '<', '>', ou '&'.

Par exemple,

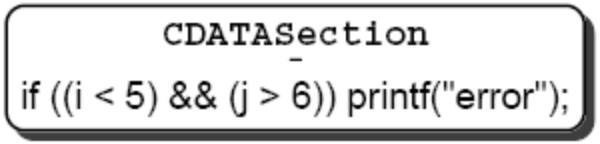
```
<?xml version='1.0'?>
<PROGRAMME>
   if ((i < 5) && (j > 6)) printf("error");
</PROGRAMME>
```

est incorrect !

Sections CDATA

Les **sections littérales** CDATA permettent d'inclure du texte qui n'est **pas analysé** par le parseur :

```
<?xml version='1.0'?>
<PROGRAMME>
   <![CDATA[if ((i < 5) && (j > 6)) printf("error"); ]]>
</PROGRAMME>
```



DTD

Les **DTD** (*Document Type Definition*) définissent la structure de l'information contenue dans les documents cibles

- Impliquent la notion de validité : accord entre les structures attendues et observées
- Parser : application permettant de d'analyser l'information contenue dans un document XML
  - `Peuvent être validateurs (ou non) s'ils vérifient la validité des documents traités`

Utilité de la DTD

Une DTD est une description de *l'interface* entre le *producteurs* et les *consommateurs* des données/documents XML :

- le producteur peut contrôler la qualité des données/documents produits
- le consommateur peut séparer la vérification syntaxique des données/documents (parseur) de la logique de l'application

Exemple de DTD

```
<!ELEMENT Officiel (#PCDATA | cinéma | film)*>
<!ELEMENT cinéma (nom, adresse, (séance)*)>
<!ELEMENT nom (#PCDATA) >
<!ELEMENT adresse (ville, rue, (numéro)?)>
<!ELEMENT séance EMPTY>
<!ATTLIST séance heure NMTOKEN #REQUIRED ref_film IDREF #REQUIRED>
<!ELEMENT film (titre, année>
<!ATTLIST film film_id ID #REQUIRED  acteurs IDREFS #IMPLIED>
<!ELEMENT titre (#PCDATA) >
<!ELEMENT année (#PCDATA) >
```

Utilisation de la DTD

On ajoute au début du document XML la clause DOCTYPE.

- Définition locale:

```
<!DOCTYPE Officiel [
<!ELEMENT Officiel (#PCDATA|cinéma|film)*>
<!ELEMENT cinéma (nom, adresse, (séance)*)>
...]>
```

- Définition externe :

```
<!DOCTYPE Officiel SYSTEM "officiel.dtd">
```

Attention : ne pas redéfinir DOCTYPE dans le fichier DTD mais directement les définitions

Déclaration du type d'Élément

Un élément est défini par un nom et un *modèle de contenu* :

```
<!ELEMENT nom type_element>
```

- *Expression régulière* sur l'alphabet des noms d'éléments ;
- *EMPTY* = élément vide;

```
<!ELEMENT elt EMPTY>
```

- *ANY* = toute combinaison de tous les éléments ;

```
<!ELEMENT elt ANY>
```

- *#PCDATA* = texte

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

- Contenu mixte

```
<!ELEMENT elt  (#PCDATA | A|B)* >
```

Définitions d'éléments

Un **cinéma** a

- un nom
- une adresse optionnelle
- une suite de séances.

```
<!ELEMENT cinéma (nom,adresse?,séance*)>
```

Une **personne** a

- un nom
- plusieurs numéros de téléphone
- au moins une adresse email

```
<!ELEMENT personne (nom,tel*,email+)>
```

Exemples de DTD (1)

```
<!ELEMENT Officiel (#PCDATA | cinéma | film)*>
<!ELEMENT cinéma (nom, adresse, (séance)*)>
<!ELEMENT nom (#PCDATA) >
<!ELEMENT adresse (ville, rue, (numéro)?)>
...
<!ELEMENT séance EMPTY>
<!ATTLIST séance heure NMTOKEN #REQUIRED ref_film IDREF #REQUIRED>
<!ELEMENT film (titre, année)>
<!ATTLIST film film_id ID #REQUIRED  acteurs IDREFS #IMPLIED>
<!ELEMENT titre (#PCDATA) >
...
```

Exemples de DTD (2)

```
(dans fichier XML)
...
<!DOCTYPE biblio[
        <!ELEMENT biblio (livre)*>
        <!ELEMENT livre (titre, auteur, nb_pages)>
        <!ELEMENT titre (#PCDATA)>
        <!ELEMENT auteur (#PCDATA)>
        <!ELEMENT nb_pages (#PCDATA)>
        <!ATTLIST livre type (roman|nouvelles|poemes|théâtre)
                        #IMPLIED lang CDATA "fr">
 ]>
...
```

Déclaration des Attributs

- Un attribut est défini par l'élément auquel il est associé, un type, un mode et éventuellement sa valeur par défaut :

```
<!ATTLIST élément nom type mode [defaut]>
```

Types d'attributs

Types acceptables pour un attribut :

- Chaînes de caractères

```
<!ATTLIST elt attr CDATA>
```

  - NMTOKEN/NMTOKENS : chaîne(s) de caractères sans blancs

- Énumérations : séquences de valeurs alternatives séparées par |

```
<!ATTLIST img format (BMP | GIF | JPEG)  "JPEG">
```

  - Attention : pas de guillemets entre les "()"

- ID: indique que l'attribut servira d'identifiant

```
<!ATTLIST elt attr ID>
```

- IDREF, IDREFS :indique que l'attribut est une référence à un élément

```
<!ATTLIST elt attr IDREF>
```

- ENTITY/ENTITIES : entité(s)
- NOTATION : notation (entités non-XML)

Mode d'attributs

Modes fixant l'usage de l'attribut :

- #REQUIRED : la valeur de l'attribut doit être définie impérativement dans le document
- #IMPLIED : la valeur est optionnelle
- #FIXED : la valeur sera toujours la valeur par défaut déclarée dans la DTD (sert de variable cachée, ou permet d'assurer des compatibilités descendantes)

Exemples de déclaration d'attributs

- Les éléments de type *séance* ont un attribut *heure* et un attribut *ref_film* :

```
<!ATTLIST séance heure NMTOKEN #REQUIRED
                ref_film IDREF #REQUIRED>
```

- Les éléments de type *film* ont un attribut *film_id* et un attribut *acteurs* :

```
<!ATTLIST film film_id ID #REQUIRED acteurs IDREFS #IMPLIED>
```

- ...on peut rajouter la langue du film :

```
<!ATTLIST film film_id ID #REQUIRED acteurs IDREFS #IMPLIED
                langue (AN|FR|AL|ES|IT) #IMPLIED>
```

- une adresse peut être personnelle ou professionnelle :

```
<!ATTLIST adresse type CDATA #IMPLIED 'Personnelle'>
```

Entités générales et entités paramètres

```
<!DOCTYPE Officiel [
<!ENTITY copyright 'Copyright B. Amann'>
<!ELEMENT Officiel (p, année) >
<!ELEMENT p (#PCDATA) >
<!ENTITY % text '#PCDATA'>
<!ELEMENT année (%text;) >
]>
<Officiel>
<p> &copyright; </p><année>2000</année>
</Officiel>
```

Les entités paramètres
(ENTITY %)
peuvent uniquement être
utilisées dans la DTD.

Entités externes

- Segmentation du document en plusieurs sous-documents
- Réutilisation de DTDs et de déclarations
- Références vers données non-XML (NOTATION)
- Adressage:
  - URL : `<!ENTITY % autre SYSTEM 'http://pariscope.fr/ext.xml' >`
  - FPI (formal public identifier) :
    `<!ENTITY % autre PUBLIC '-//CNAM//Texte libre//FR'>`

Entités non-XML - NOTATION

Utilisation :

- déclaration du format (type = application) pour entités non-XML
- référence à une entité de type notation seulement possible comme valeur d'attribut

```
<!DOCTYPE exemple [
<!NOTATION gif SYSTEM '/usr/local/bin/xv' >
<!ENTITY myphoto SYSTEM './moi.gif' NDATA gif >
<!ELEMENT person EMPTY >
<!ATTLIST person photo NOTATION (gif) #IMPLIED>
]>
<person photo='myphoto' >
```

Alternatives aux DTD

Inconvénients des DTD :

- l'élément racine n'est pas spécifié pas dans la DTD ; 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 pas être contraint précisément, puisque l'on ne dispose que des quantifieurs ?, * et +
  - `on aimerait pouvoir dire qu'un élément doit apparaître plus de 2 fois mais toujours moins de 5...`
- on ne dispose pas de types pour les contenus des attributs et des éléments (nom, date, code postal, url, adresse mail...) ;
- 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 !

D'autres propositions ont été faites permettant de spécifier un langage XML : *XML Schema* ou *Relax NG*.

Exemple de XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>
   <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
     <xsd:element name="contacts" type="typeContacts"/>
     <xsd:element name="remarque" type="xsd:string"/>

  <!-- déclarations de types ici -->
     <xsd:complexType name="typeContacts">
<!-- déclarations du modèle de contenu ici -->
       <xsd:attribute name="maj" type="xsd:date">
     </xsd:complexType>
   </xsd:schema>
```

Espaces de noms

- Définissent des vocabulaires séparés sous XML
- Permettent la cohabitation de plusieurs domaines XML en évitant les conflits entre termes identiques
- Principes
  - un schéma (DTD) définit son propre espace de nom, dans lequel tous les noms d'éléments et d'attributs sont uniques
  - on dispose d'un mécanisme pour
    - identifier les espaces de nom utilisés dans un document
    - identifier pour chaque élément ou attribut à quel espace de nom il appartient
  - Et donc
    - toute référence à un nom d'élément est non ambiguë
    - un document peut contenir des élément définis dans plusieurs espaces de nom

Exemple d'espace de noms…

```
<html:html xmlns:html="http://www.w3.org/1999/xhtml">
<html:head>
      <html:title>Ma page</title>
      .../...
</html:head>
.../...
</html:html>
```

Exemple d'espace de noms

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <title>Ma page</title>
      .../...
</head>
.../...
</html>
```

Exemple d'espaces de noms

```
<html:html xmlns:html="http://www.w3.org/1999/xhtml"
           xmlns:mml="http://www.w3.org/1998/Math/MathML">
  <html:head>
    <html:title>Espaces de noms</html:title>
  </html:head>
  <html:body>
    ...
    <mml:math>
      <mml:apply>
        <mml:eq/>
        ...
      </mml:apply>
    </mml:math>
    ...
  </html:body>
</html:html>
```