M2 RS

Module : Services Web

Chapitre 02:

XML (eXtensible Markup Language)

Sommaire

1. Introduction — Structuration des données : quels besoins ?

→ Besoin d'échanger des données entre acteurs :

- Données bancaires
- Données de remboursement de santé
- Description d'une recherche d'itinéraire dans Google Maps
- La réponse associée: description d'un itinéraire
- ...

→ D'où la nécessité d'un format de structuration universel :

- Permettant de représenter toutes sortes de données structurées (non limitant à un certain type de données)
- Lisible sur tous les systèmes
- Le plus simple possible

→ Deux formats d'échange sont maintenant standards sur le web:

- XML (eXtensible Markup Language) - 1998
  - Généralisation de HTML pour représenter toutes sortes de données et pas seulement des pages web
  - Origine : W3C avec un soutien industriel fort (Microsoft)
- JSON (JavaScript Object Notation) - 2006
  - Sous-ensemble de JavaScript
  - Origine : développement propriétaire standardisé ensuite

2. XML : Définition

→ XML est un langage de structuration de document.

→ Un langage orienté texte et formé de balises qui permettent d'organiser les données de manière structurée.

→ Il s'est imposé comme un standard incontournable de l'informatique.

→ XML est ***extensible***: on peut créer autant de balises que l'on souhaite.

- Les balises HTML servent à formater les informations. (Les balises sont prédéfinies)
- Les balises XML ne servent qu'à structurer les documents. (le concepteur définit les balises qui ont du sens pour lui, c.à.d. son propre jeu de balises).

→ XML est ***rigoureux*** : Un fichier XML doit respecter certaines règles.

→ Un document XML ne possède aucune information sur la manière dont il doit être affiché sur un écran d'ordinateur.

→ Il est aussi bien utilisé pour le stockage de documents que pour la transmission de données entre applications.

Remarques:

- Le langage XML dérive de SGML (Standard Generalized Markup Language) et de HTML (HyperText Markup Language).
- Un fichier XML est un simple fichier texte, contenant des balises.
- XML n'est pas un langage à proprement parler comme peut l'être HTML, XML est une famille de langages (est un méta-langage) adaptés aux types d'informations à décrire.
- Sa simplicité, sa flexibilité et ses possibilités d'extension ont permis de l'adapter à de multiples domaines allant des données géographiques au dessin vectoriel en passant par les échanges commerciaux.

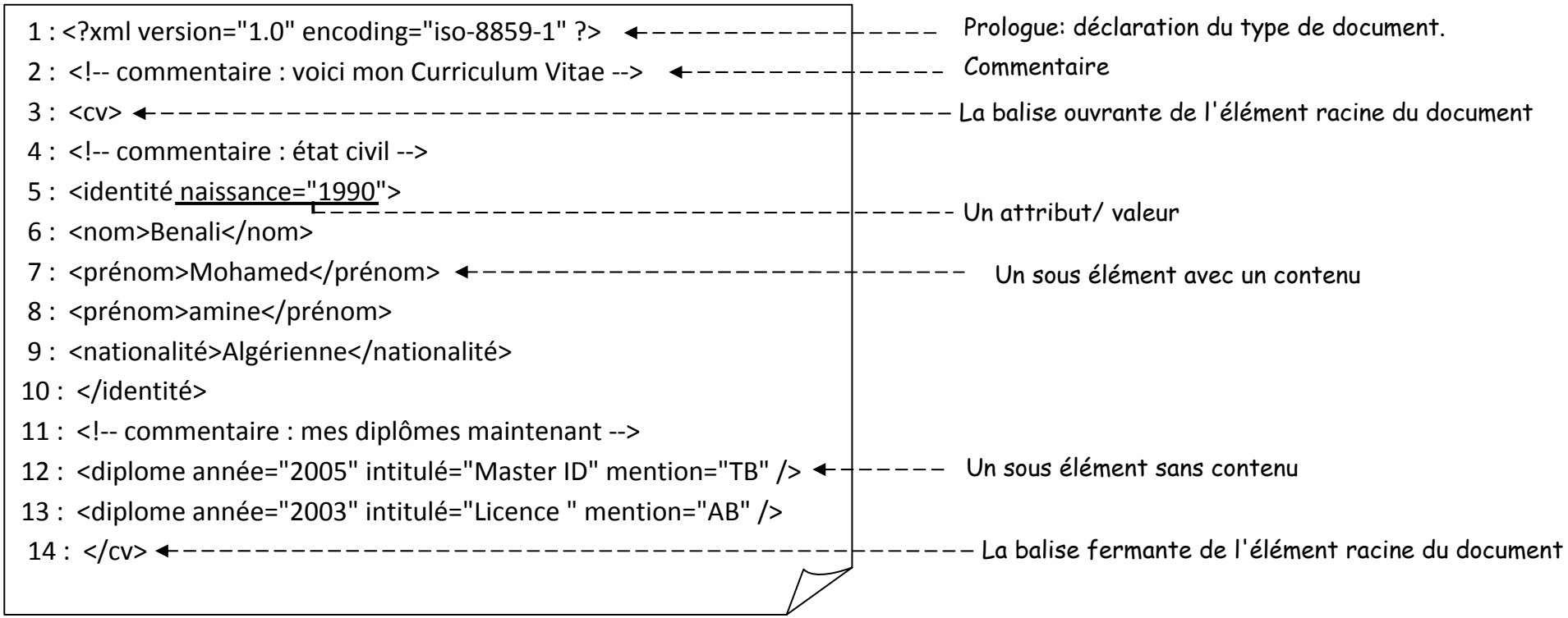
Avantages

- ✓ Utilise du texte (peut être lu et écrit directement par des humains ou par des applications)
- ✓ Fondé sur une syntaxe de type « texte balisé» simple qui a fait le succès de HTML
- ✓ Technologie universelle pour structurer l'information
- ✓ Adapté à la diffusion et à l'échange d'information (format d'échange universel)
- ✓ Orienté vers la structuration des données
- ✓ Indépendant des plates-formes, et des systèmes d'exploitation

Objectif :

- ✓ Faciliter les échanges de données entre les machines.
- ✓ Décrire les données de manière aussi bien compréhensible par les hommes qui écrivent les documents XML que par les machines qui les exploitent.

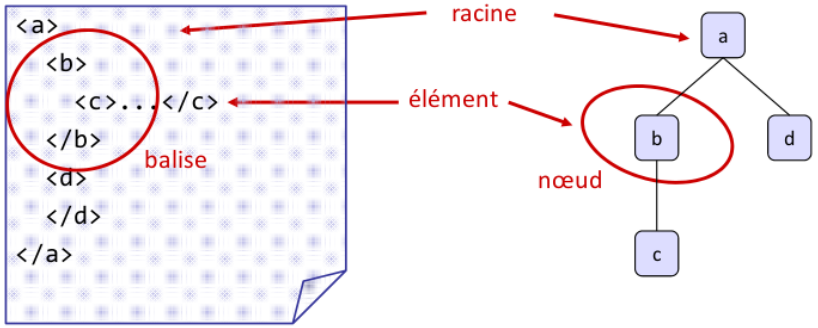
Exemple d'un document XML :



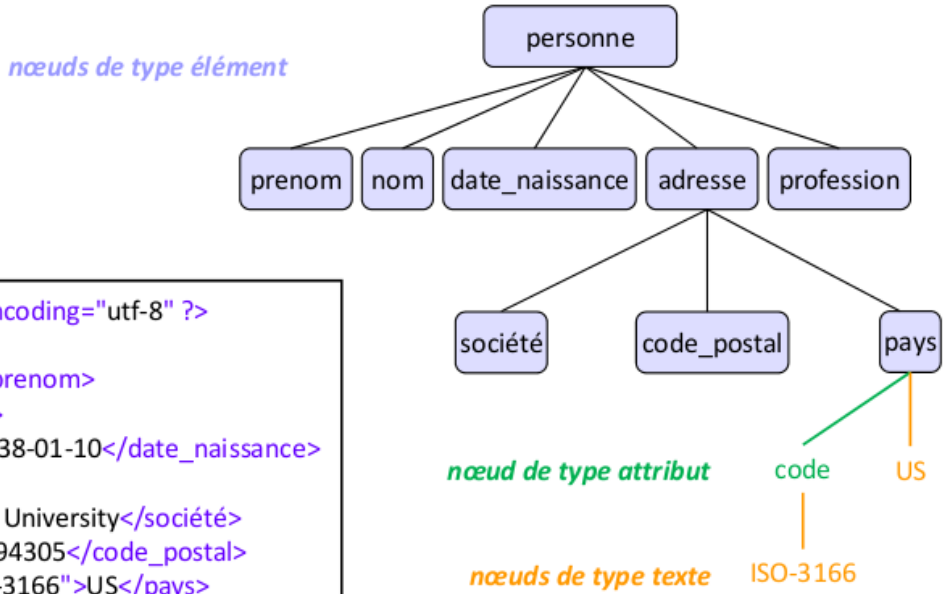
monCV.xml

Dualité texte balisé - arbre: la représentation graphique

Un texte XML essentiellement représente un arbre



Exemple : carte de visite




```
<?xml version="1.0" encoding="utf-8" ?>
<personne>
  <prenom>Donald</prenom>
  <nom>Knuth</nom>
  <date_naissance>1938-01-10</date_naissance>
  <adresse>
    <société>Stanford University</société>
    <code_postal>CA 94305</code_postal>
    <pays code = "ISO-3166">US</pays>
  </adresse>
  <profession>informaticien</profession>
</personne>
```

3. Le XML et le HTML

Différences HTML / XML

```
<!DOCTYPE html>
<html>
<head>
<meta charset ="utf-8">
<title> Mohamed Benali </title>
</head>
<body>
<p> Mohamed Benali </p>
<address>
Université de Jijel<br>
BP98<br>
Ouled Aïssa<br>
18000 Jijel
</address>
</body>
</html>
```

Mohamed Benali,
Université de Jijel
BP 98 Ouled Aïssa 18000 Jijel .

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<enseignant corps="maıtre de conférences">
<prenom>Mohamed</prenom>
<nom>Benali </nom>
<adresse>
<structure>Université de Jijel</structure>
<bp>BP98</bp>
<rue>Ouled Aïssa</rue>
<cp>18000</cp>
<ville>Jijel</ville>
</adresse>
</enseignant>
```

HTML : (afficher des données)

- → description d'un document (sur le modèle d'un article imprimé)
- → structure : titres , paragraphes, listes, …
- → présentation : italiques, gras, …
- → insertion d'images, vidéos, …
- → perte du sens

XML : (décrit, structure, échange des données)

- → structuration logique de données
- → pas de représentation envisagée a priori
- → on crée ses propres balises (extensibilité du langage)

4. La syntaxe du XML

Un document XML est composé de plusieurs parties :

- Entête de document précisant la version et l'encodage,
- Des règles optionnelles permettant de vérifier si le document est valide,
- Un arbre d'*éléments* basé sur un élément appelé *racine*
  - Un *élément* possède un *nom*, des *attributs* et un *contenu*
  - Le contenu d'un élément est :
    * du texte
    * d'autres éléments (les éléments enfants).
  - Un élément est marqué par une *balise ouvrante* et une *balise fermante*.
    * une balise ouvrante est notée <nom attributs...>
    * une balise fermante est notée </nom>
- Les commentaires

Règles d'écriture au format XML :

Les règles que doit suivre un document XML sont les suivantes (si toutes ces contraintes sont respectées, le document XML est dit *bien formé*) :

1) La première ligne représente le Prologue, elle doit être de la forme

   <?xml version="1.0" encoding="iso-8859-1" ?> ;

   les deux attributs spécifient la version de XML utilisée (1.0 ou 1.1 qui sont très similaires) et le codage des caractères (utf-8 par défaut).
2) Les éléments définissent la structure du document (arbre d'éléments). Un élément est délimité par :

   — une balise ouvrante <nom attributs...>
   — une balise fermante </nom> obligatoire.

   par exemple : <balise>contenu</balise>

   Le contenu de l'élément se trouve entre les deux balises. Ce sont des éléments enfants et/ou du texte. Dans le cas où l'élément n'a pas de contenu (élément vide), on peut regrouper ses deux balises en une seule <nom attributs.../>
3) Les attributs donnent des précisions sur les éléments et leurs contenus. Ils sont placés dans la balise ouvrante. toujours donner une valeur aux attributs, en suivant la syntaxe<balise attr="val"> (les guillemets sont obligatoires, les attributs ne sont pas répétés dans la balise fermante).

4) Certains caractères sont interdits dans le contenu des éléments. Comme il est interdit de les employer, XML propose une représentation appelée *entité.* Il est possible de créer ses propres entités. Dans ce cas, elles peuvent représenter beaucoup plus qu'un caractère, toute une chaîne, voire même tout un arbre XML. les entités sont systématiquement de la forme *&nom*.
5) Fermer toutes les balises ouvertes, une balise sans contenu pourra être ouverte et immédiatement fermée en faisant suivre son nom d'un slash, par exemple avec la balise br(passage à la ligne en HTML) :
 .
6) Vérifier à l'ordre de fermeture des balises : la première ouverte est toujours la dernière fermée.
7) Respecter la casse : on peut utiliser majuscules et minuscules dans les noms de balises mais une fois qu'un nom d'élément a été fixé, il faut s'y tenir, la balise <cv> ne pourra être fermée ni par </Cv> ni par </CV>.
8) Ne pas utiliser de caractères réservés à XML dans le texte du document : <, > et & ; ces caractères pourront être respectivement obtenues à l'aide des entités <, > et&amp.
9) Les noms d'éléments et d'attributs doivent être des *noms XML* :
   - Le premier caractère est une lettre quelconque ou un _ (underscore ou tiret bas) ;
   - Les caractères suivants peuvent être des lettres, des chiffres, des tirets bas (_), des traits d'union (-) ou des points (.).
   - Il n'y a pas de limitation sur la longueur d'un nom XML.
   - Les espaces non autorisés

Remarques:

- Après le prologue, on peut trouver plusieurs parties optionnelles délimitées par <?...?> ou <!...>.

  Ce sont des instructions de traitement, des directives pour l'analyseur XML.

- Choses interdites :
  - ☒ Plusieurs racines dans le document,
  - ☒ Eléments non terminés (NB: XML est sensible à la casse),
  - ☒ Eléments qui se chevauchent.

Exercice :

```
1. <?xml version="1.0"?>
2. <!-- this is a note -->
3. <note date=3 janvier>
4. <to>Bob</To>
5. <from>Alice</from>
6. <heading>Reminder</heading>
7. <body>Don't forget me this weekend!</body>
8. </note>
9. <note date="5 janvier" <!-- this is another note --> >
10. <to>Alice</to>
11. <from>Bob
12. <body>No problem & see you soon</body>
13. </note>
14. <note />
```

Questions :

1. Ce document est-il bien formé (i.e. respecte-t-il la syntaxe XML) ?
2. S'il ne l'est pas, corrigez les erreurs.

5. Validation d'un document XML

→ La correction permet de vérifier *le vocabulaire* et *la structure* d'un document XML avant de commencer à le traiter.

→ Il y a deux niveaux de correction pour un document XML :

1) Un document XML *bien formé* (*well formed*) respecte les règles syntaxiques d'écriture XML
   - Ecriture des balises, imbrication des éléments, entités, etc. C'est le niveau de base de la validation.
2) Un document XML *valide* respecte des règles supplémentaires sur les noms, attributs et organisation des éléments.
   - Validation des contraintes plus précises et propres à un langage XML (définies par le créateur).

→ La validation va permettre de répondre aux questions suivantes :

- Quels sont les noms des éléments que je souhaite exploiter au sein de mon document XML ?
- Quels sont les attributs que je souhaite associer à un élément ?
- Un attribut donné, est-il obligatoire ou facultatif dans un élément ?
- Est-ce qu'un élément imbrique d'autres éléments, et si oui lesquels ?
- Est-ce qu'un élément est toujours vide ou non ?

→ Pour contrôler la validité d'un document XML, il faut disposer d'un fichier contenant des règles.

→ Il existe plusieurs langages à divers niveaux d'expressivité pour modéliser des règles de validité plus ou moins précises et d'une manière plus ou moins lisible :

- ***DTD (Document Type Definition)***
  - Principalement, définition d'un vocabulaire : Noms des balises et attributs
- ***XML Schema***
  - Vérification plus poussée de la structure
  - Notion de type de données

→ La validation indique que soit le document est valide, soit il contient des erreurs comme :

- tel attribut de tel élément contient une valeur interdite par telle contrainte,
- il manque tel sous-élément dans tel élément, etc.

Les avantages:

☑ Faciliter l'échange et la mise en commun de documents produits par des rédacteurs différents.

☑ Aider les développeurs qui conçoivent des outils automatiques pour traiter les documents respectant les mêmes règles de validité.

6. DTD (Document Type Definition)

→ Le DTD est l'ensemble des règles et des propriétés que doit suivre le document XML.

→ Ces règles définissent généralement :

→ le nom et le contenu de chaque élément (balise)

→ le contexte dans lequel ils doivent exister.

→ Cette formalisation des éléments est particulièrement utile lorsqu'on utilise de façon récurrente des balises dans un document XML.

→ Cette spécification d'une grammaire pour un langage et la possibilité de tester automatiquement son respect par un document donné, présentent les avantages suivants :

- ✓ Faciliter l'échange et la mise en commun de documents produits par des rédacteurs différents. Autrement dit, plusieurs concepteurs peuvent se mettre d'accord pour utiliser un DTD commun pour échanger leurs données.
- ✓ Aider les développeurs qui conçoivent des outils automatiques pour traiter les documents respectant la même DTD.

Liaison entre un document XML et une DTD : *Comment intégrer une DTD ?*

➥ *DTD Interne (au document XML)*

- La DTD arrive dans l'entête du document XML juste après le prologue et avant la racine.
- On trouve d'abord le mot-clé DOCTYPE suivi de l'élément servant de racine au document et enfin la DTD elle-même entre crochets.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE cv [ . . . ]>
<cv>
.  .  .
</cv>
```

➥ *DTD Externe*

- La DTD est détachée dans un fichier séparé (un fichier .dtd), on se contente d'y faire référence dans l'entête du document XML.
- On retrouve le mot-clé DOCTYPE suivi de l'élément servant de racine, puis le mot-clé SYSTEM suivi d'une URI menant au fichier DTD.

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE cv SYSTEM "cv.dtd">
<cv>
. . .
</cv>
```

➥ *DTD Mixte*

- Mélange de deux notations pour avoir une partie de la DTD dans un fichier séparé et une autre partie embarquée dans le document XML :

```
<?xml version="1.0" encoding=" utf-8" standalone="no" ?>
<!DOCTYPE cv SYSTEM "cv.dtd" [ . . . ]>
 <cv>
 . . .
</cv>
```

Syntaxe et contenu d'une DTD

Syntaxe :

- Syntaxe particulière, mais ressemblant au reste de XML
- Quelques balises prédéfinies de la forme <!BALISE paramètres>
- Pas de déclaration XML dans un fichier .dtd

Contenu principal :

a. Déclaration de chaque élément (<!ELEMENT ...>): son nom et le contenu autorisé

b. Déclaration des attributs possibles d'un élément (<!ATTLIST ...>): nom et options

c. Déclaration des entités (<!ENTITY ... >) : son nom et le texte associé

Déclaration d'éléments

Déclaration d'attributs

Déclaration d'entités

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ATTLIST body lang CDATA #IMPLIED! signature CDATA #REQUIRED>
<!ENTITY sl "Mohamed benmohamed">
...
```

a) Définir les Eléments et leurs Contenus

→ Déclaration des éléments à apparaître dans le document, ainsi que leurs imbrications possibles. La forme générale est la suivante :

```
<!ELEMENT nom_element modèle_de_contenu>
```

→ Les différents modèles de contenu utilisables dans les DTD :

- *Contenu Purement Textuel :*

  Si l'élément peut contenir du texte brut mais pas de nouvelles balises, on utilisera le modèle de contenu PCDATA (Parsed Character Data). Exemple : `<!ELEMENT téléphone (#PCDATA)>`

  Aucune balise n'est donc tolérée dans ce type de contenu mais, par contre, il est possible d'y utiliser des entités.

- *Contenu Vide :*

  Un élément peut également n'avoir aucun contenu, Cela se précise dans la DTD de la manière suivante : `<!ELEMENT nomElément EMPTY>`. Une telle balise sera donc ouverte et immédiatement refermée avec la notation suivante : `< nomElément />`.

- *Sous-éléments*

  Un élément peut contenir une suite ordonnée de sous-éléments :

```
<!ELEMENT nomElément (Fils1,Fils2, … )>
```

Exemple :

```
<!ELEMENT identité (prénom,nom)>
```

indique que l'élément `identité` doit contenir, *dans l'ordre*, un élément `prénom`, un élément `nom`, et rien d'autre.

Il est possible de moduler le nombre d'apparitions d'un sous-élément en utilisant des quantifieurs après les noms d'éléments. Les quantifieurs utilisables dans les DTD sont :

- `?` : 0 ou 1 fois ;
- `*` : 0 ou plus ;
- `+` : 1 ou plus.

L'exemple :

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

Indique que l'élément `identité` doit contenir, toujours en respectant l'ordre, un ou plusieurs éléments `prénom`, un élément `surnom` facultatif et exactement un élément `nom` :

- ***Alternatives***

Définir les sous-éléments qui peuvent apparaître de manière exclusive.

```
<!ELEMENT nomElément (fils1+|fils2*)>
```

Exemple : `<!ELEMENT expérience (stage|emploi)>`

- une `expérience` professionnelle peut être soit un `emploi`, soit un `stage`

Combinaisons

Combiner les syntaxes vues précédemment.

Exemple :

```
<!ELEMENT diplôme ( (intitulé,année) | (année,compétences,stage?)+ )>
```

Dans ce cas, un `diplôme` est :

→ soit l'`intitulé` du diplôme et l'`année` d'obtention
→ soit une suite d'`année`, avec les `compétences` acquises cette année là, éventuellement validées par un `stage`.

Contenu Quelconque

Autoriser n'importe quel contenu à apparaître dans un élément.

```
<!ELEMENT nomElément ANY>
```

Dans ce cas, l'élément pourra contenir du texte brut mélangé avec toute balise définie dans la DTD, dans n'importe quel ordre, autant de fois que l'on veut.

Exemple : Une DTD que l'on enregistre dans un fichier nommé `livres.dtd` :

```
<!ELEMENT liste_livres (livre+)>
<!ELEMENT livre (titre,auteur+,éditeur,description?,prix)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT éditeur (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT prix (#PCDATA)>
```

et un fichier XML la respectant :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<!DOCTYPE liste_livres SYSTEM "livres.dtd">
<liste_livres>
<livre>
<titre>Comprendre XSLT</titre>
<auteur>Bernd Amann</auteur>
<auteur>Philippe Rigaux</auteur>
<éditeur>OReilly</éditeur>
<description> Le livre suit une double démarche de présentation des aspects les
plus simples, puis, progressivement, les plus complexes du langage XSLT, et
d'application des mécanismes de ce langage à des cas concrets d'utilisation.
</description>
<prix>33 euros</prix>
</livre>
<livre>
<titre>Learning XML</titre>
<auteur>Erik T. Ray</auteur>
<éditeur>OReilly</éditeur>
<prix>40 dollars</prix>
</livre>
</liste_livres>
```

b) Définir les Attributs

→ Syntaxe : le mot ATTLIST, suivi du nom de l'élément concerné, suivi de la liste de ses attributs : pour chacun, on trouvera son *nom*, son *type* et son *caractère optionnel ou non*.

→ Une déclaration d'attributs typique aura la forme suivante :

```
<!ATTLIST élément attribut TypeAttribut TypeDeDefinition ValeurParDefaut>
```

Exemple :
```
<!ATTLIST identité prénom CDATA #REQUIRED
                   nom CDATA #REQUIRED
                   surnom CDATA #IMPLIED>
```

→ Il y a trois types de définition :

- #REQUIRED : l'attribut DOIT avoir une valeur, l'attribut est donc obligatoire.
- #IMPLIED : l'attribut est facultatif.
- #FIXED : l'attribut a une valeur fixe, définie à l'avance.

→ Les différents types d'attributs.

↳ *CDATA*

C'est le type le plus général, il permet de saisir un texte quelconque pour un attribut de ce type.

Exemple : `<!ATTLIST dossier chemin CDATA #IMPLIED>`

- *ÉNUMERATION*

Dans ce cas, l'attribut ne peut prendre qu'un nombre fini de valeurs et l'on en donne la liste exhaustive, ces valeurs étant séparées par des |.

Exemple:

```
<!ATTLIST date jour (lundi|mardi|mercredi|jeudi|vendredi|samedi|dimanche) #REQUIRED >
```

- *ID*

Identifiant XML unique dans le document (identifie de manière unique l'élément)

Exemple: `<!ATTLIST dossier reference ID #REQUIRED>`

- *NMTOKEN et NMTOKENS*

Il s'agit ici un nom XML mais sans restriction sur le premier caractère (qui peut donc être un chiffre). Une contrainte essentielle est donc qu'un attribut de ce type ne contiendra pas d'espace. Exemple - code postal :

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

NMTOKENS : Une suite de NMTOKEN, séparés par des espaces.

c) Définir les Entités

→ Définir des raccourcis (ou des alias) qui seront utilisables dans les documents XML liés à la DTD.

- Ceci évite de répéter plusieurs fois le même texte

```
<!ENTITY sl "Mohamed benmohamed">
```

→ Certaines entités sont déjà définies en XML : < (<), > (>),& (&), " ("), et ' (').

DTD Insuffisant ?

DTD permet d'exprimer des contraintes assez basiques

- ✓ Liste des éléments et de leurs attributs
- ✓ Règles de structuration des éléments

Mais :

- × Impossible de typer réellement les attributs (un typage très limité des données).
- × Le nombre d'apparitions d'un élément ne peut pas être contraint précisément, puisque l'on ne dispose que des quantifieurs ?, * et +,… .
- × Il ne peut y avoir deux éléments de même nom dans deux contextes différents.
- × Pas d'héritage possible entre éléments : notation lourde.
- × Pas de prise en compte des espaces de noms.
- × le langage utilisé pour définir une DTD n'est pas un langage XML.

→ Pour palier à ces manques, d'autres langages de schéma plus complets ont été faits, par exemple les XML Schema.

7. XML Schéma

→ **XML Schema** publié comme recommandation par le W3C en mai 2001 est un langage de description de format de document **XML** permettant de définir la structure et le type de contenu d'un document **XML**. Cette définition permet notamment de vérifier la validité de ce document.

→ Définit les balises et leurs attributs

→ Définit les contraintes de structure des documents

→ Espace de noms = préfixe permettant d'éliminer les conflits lorsque plusieurs balises ont des noms identiques, URL (fictive) utilisée comme identifiant

→ Les **Schémas XML** offrent plus de possibilités que les DTD.

- Utilisation et définition de types, contraintes sur les contenus.
- Possibilité de définir précisément le nombre d'apparitions d'un élément.
- Espaces de noms supportés.
- Format XML, parsable facilement.

→ Les **Schémas XML** s'écrivent à l'aide d'un langage de type XML.

→ Un fichier dans lequel est écrit un **Schéma XML** porte l'extension **".xsd"**.

Apports des schémas XML

- ✓ Les **Schémas XML** permettent tout d'abord de *typer* les données. même, il est possible d'aller plus loin en créant nos propres types de données.
- ✓ **Les contraintes :** les **Schémas XML** permettent d'être beaucoup plus précis que les DTD lors de l'écriture des différentes contraintes qui régissent un document XML.
- ✓ **Des définitions XML :** Un des principaux avantages des **Schémas XML** est qu'ils s'écrivent grâce au XML. Ainsi, pour exploiter un document XML et le Schéma qui lui est associé, vous n'avez en théorie plus besoin de plusieurs outils.

Exemple de schéma pour se faire une idée :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="reference" type="ElemReference" />
  <xsd:complexType name="ElemReference">
     <xsd:sequence>
        <xsd:element name="titre" type="xsd:string" />
        <xsd:element name="auteur" type="xsd:string" />
        <xsd:element name="ISBN" type="xsd:string" />
     </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Remarque :

> La syntaxe est moins lisible que celle des DTD, car ils sont écrits en XML.

Structure d'un schéma XML

→ Bien que les Schémas XML soient écrits avec un langage de type XML, le fichier n'a pas cette extension.

→ Un fichier dans lequel est écrit un Schéma XML porte l'extension "**.xsd**". (Le schéma peut être aussi intégré au fichier XML lui-même).

→ La première ligne d'un Schéma XML est :

```
<?xml version="1.0" encoding="UTF-8" ?>
```

→ Comme pour un fichier XML classique, le **corps d'un Schéma XML** est constitué d'un ensemble de **balises**.

→ Cependant, une chose ne change pas : la présence d'un **élément racine**, c'est-à-dire la présence d'une balise qui contient toutes les autres. Mais, contrairement à un fichier XML, son nom est imposé : `<xsd:schema />`

Exemple :

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

```
<personne>
      <prenom>Gaston</prenom>
      <nom>Lagaffe</nom>
</personne>
```

Liaison entre un document XML et un Schéma XML :

➥ Pour attribuer un schéma de validation local à un document XML, on peut ajouter un attribut situé dans un *namespace* spécifique :

```
<?xml version="1.0"?>
<reference
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="reference.xsd">
<titre>Comprendre XML</titre>
<auteur>Bernd Amann et Philippe Rigaux</auteur>
<ISBN>2-84177-148-2</ISBN>
</reference>
```

➥ Lorsque le schéma est public, mis sur un serveur, c'est un peu différent car il faut définir un *namespace* et l'URL d'accès, et il faut ajouter les attributs xmlns et targetNamespace valant le même namespace à la racine du schéma :

```
<?xml version="1.0"?>
<reference
xmlns="http://www.iut-lannion.fr"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.iut-lannion.fr reference.xsd">
<titre>Comprendre XML </titre>
<auteur>Bernd Amann et Philippe Rigaux</auteur>
<ISBN>2-84177-148-2</ISBN>
</reference>
```

Principes généraux des Schémas XML

→ Comme une DTD, un schéma permet de définir des éléments, leurs attributs et leurs contenus. Mais il y a une notion de typage beaucoup plus forte qu'avec une DTD. Avec un schéma, il faut définir les types de données très précisément :

- ***Nature des données :*** chaîne, nombre, date, etc.
- ***Les contraintes qui portent dessus*** : domaine de définition, expression régulière, etc.

→ Avec ces types, on définit les éléments :

- noms et types des attributs,
- sous-éléments possibles avec leurs répétitions,
- les alternatives,
- etc.

} C'est tout cela qui complique beaucoup la lecture d'un schéma

Structure générale d'un schéma

Un schéma est contenu dans un arbre XML de racine <xsd:schema>. Le contenu du schéma définit les éléments qu'on peut trouver dans le document. Voici un squelette de schéma :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="itineraire" type="TypeItineraire" />
    ... définition du type TypeItineraire ...
</xsd:schema>
```

Il valide le document partiel suivant :

```
<?xml version="1.0"?>
<itineraire>
...
</itineraire>
```

Définition d'éléments

Un élément <*nom*>*contenu*</*nom*> *du document est défini par un* élément :

```
<xsd:element name="nom" type="TypeContenu">
```

dans le schéma.

Dans l'exemple suivant, le type est `xsd:string`, c'est du texte quelconque :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="message" type="xsd:string"/>
</xsd:schema>
```

Ce schéma valide le document suivant :

```
<?xml version="1.0"?>
<message>Tout va bien !</message>
```

Types de données

L'exemple précédent indique que l'élément `<message>` doit avoir un contenu de type `xsd:string`, c'est à dire du texte.

Ce type est un « type simple ». Il y a de nombreux types simples prédéfinis, dont :

- **chaîne** :
  - `xsd:string` est le type le plus général
  - `xsd:token` vérifie que c'est une chaîne nettoyée des sauts de lignes et espaces d'indentation
- **date et heure :**
  - `xsd:date` correspond à une chaîne au format AAAA-MM-JJ
  - `xsd:time` correspond à HH:MM:SS.s
  - `xsd:datetime` valide AAAA-MM-JJTHH:MM:SS, on doit mettre un T entre la date et l'heure.
- **Nombres :**
  - `xsd:decimal` valide un nombre réel
  - `xsd:integer` valide un entier
  - `xsd:nonNegativeInteger`
  - `xsd:positiveInteger`
- **booléenne**
  - `xsd:boolean` permet de n'accepter que true, false, 1 et 0 comme valeurs dans le document.
- **autres :**
  - `xsd:ID` pour une chaîne identifiante,
  - `xsd:anyURI` pour valider des URI (URL ou URN).

Restrictions sur les types

Lorsque les types ne sont pas suffisamment contraints et risquent de laisser passer des données fausses, on peut rajouter des contraintes.

Elles sont appelées *facettes (facets).*

Dans ce cas, on doit définir un type `simpleType` et lui ajouter des restrictions.

Voici un exemple :

```
<xsd:element name="temperature" type="TypeTemperature" />

<xsd:simpleType name="TypeTemperature">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="-30"/>
    <xsd:maxInclusive value="+40.0"/>
  </xsd:restriction>
</xsd:simpleType>
```

Définition de restrictions

La structure d'une restriction est :

```
<xsd:restriction base="type de base">
    <xsd:CONTRAINTE value="PARAMETRE"/>
    ...
</xsd:restriction>
```

Restriction communes à tous les types

Ces facettes sont communes à tous les types.

Longueur de la donnée :

- `xsd:length,`
- `xsd:maxLength,`
- `xsd:minLength,`
- …

Ces contraintes vérifient que la donnée présente dans le document a la bonne longueur.

Énumération de valeurs possibles :

```
<xsd:simpleType name="TypeFreinsVélo">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="disque"/>
    <xsd:enumeration value="patins"/>
    <xsd:enumeration value="rétropédalage"/>
  </xsd:restriction>
</xsd:simpleType>
```

Type complexe

Pour modéliser un élément <persone> ayant deux éléments enfants <prénom> et <nom>, il suffit d'écrire ceci :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="personne" type="TypePersonne"/>
    <xsd:complexType name="TypePersonne">
        <xsd:all>
            <xsd:element name="prénom" type="xsd:string"/>
            <xsd:element name="nom" type="xsd:string"/>
        </xsd:all>
    </xsd:complexType>
</xsd:schema>
```

La structure <xsd:all> contient une liste d'éléments qui doivent se trouver dans le document à valider. Il y a d'autres structures.

Contenu d'un type complexe

On s'intéresse à ce qu'on met dans un `<xsd:complexType>`

```
<xsd:complexType name="TypePersonne">
    <xsd:sequence> ou <xsd:choice> ou <xsd:all>...
</xsd:complexType>
```

Les enfants peuvent être :

- `<xsd:sequence> éléments. . . </xsd:sequence>` :

  ces éléments doivent arriver dans le même ordre

- `<xsd:choice> éléments. . . </xsd:choice>` :

  le document à valider doit contenir l'un des éléments

- `<xsd:all> éléments. . . </xsd:all>` :

  le document à valider doit contenir certains de ces éléments et dans l'ordre qu'on veut.

Exemple de séquence

→ Pour représenter une adresse, les éléments `<destinataire>`, `<rue>` et `<cpville>` doivent se suivre dans cet ordre :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="adresse" type="TypeAdresse"/>
    <xsd:complexType name="TypeAdresse">
        <xsd:sequence>
            <xsd:element name="destinataire" type="xsd:string"/>
            <xsd:element name="rue" type="xsd:string"/>
            <xsd:element name="cpville" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Nombre de répétitions

→ Dans le cas de la structure `<xsd:sequence>`, il est possible de spécifier un nombre de répétition pour chaque sous-élément.

```
<xsd:complexType name="TypePersonne">
    <xsd:sequence>
        <xsd:element name="prénom" type="xsd:string"
                minOccurs="1" maxOccurs="2"/>
        <xsd:element name="nom" type="xsd:string"
                minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
```

→ Par défaut, les nombres de répétitions min et max sont 1.

→ Pour enlever une limite sur le nombre maximal, il faut écrire `maxOccurs="unbounded"`.

Définition d'attributs

Les attributs se déclarent dans un `<xsd:complexType>` en utilisant la balise `<xs:attribute />`:

```
<xsd:complexType>
…
<xs:attribute name="score" type="xs:integer" use="required" default="0" />
…
</xsd:complexType>
```

- ☑ Indiquer le nom de l'attribut avec l'attribut `<xs:attribute name="score" />`
- ☑ Définir le type du contenu de l'élément en utilisant l'attribut `type`
- ☑ Préciser son caractère obligatoire ou optionnel (`required` ou `optional`) à l'aide de l'attribut `use` ;
- ☑ Éventuellement, indiquer une valeur par défaut avec l'attribut `default`.