Fiche XML
Mise en forme Babillon Damien avec les notes d'Elias Harrous. ING1 2k15, ©, rédigé en LaTeX.

1 Le langage XML et DTD

1.1 XML

Le concept XML vise essentiellement
- à séparer l'information de sa présentation
- à pouvoir accéder facilement à toute ou partie de cette information
- structurer l'information sous forme d'arbre

Structure d'un document XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<personne nom="toto" prenom="titi" >
   <adresse>2 bd Lucien Favre</adresse>
   <abonne value="true" />
</personne>
```

La norme XML définit une définition de document type appelée DTD (Document Type Definition)

1.2 Document Type Definition

Une DTD peut être définie de 2 façons :
- sous forme interne, c'est-à-dire en incluant la grammaire au sein même du document
- sous forme externe, soit en appelant un fichier contenant la grammaire à partir d'un fichier local ou bien en y accédant par son URL

Dans une DTD, on utilise le mot clé ELEMENT pour définir la structure d'un nœud :
< ! ELEMENT Nom Modèle >, avec Modèle = Any, Empty, ...

ANY	L'élément peut contenir tout type de données en caractères et les autres éléments de votre choix définis dans votre DTD
EMPTY	L'élément ne contient pas de données spécifiques HTML classique par exemple ou <img />
#PCDATA	L'élément doit contenir une chaîne de caractères

Multiplicité de l'élément :

Opérateur	Signification	Exemple
+	L'élément doit être présent au minimum une fois	A+
*	L'élément peut être présent plusieurs fois (ou aucune)	A*
?	L'élément peut être optionnellement présent	A ?
|	L'élément A ou l'élément B peuvent être présents	A|B
,	L'élément A doit être présent et suivi de l'élément B	A,B
()	Les parenthèses permettent de regrouper des éléments afin de leur appliquer les autres opérateurs	(A,B)+

Dans une DTD, on utilise le mot clé ATTLIST pour définir les attributs d'un nœud : < ! ATTLIST Elément Attribut Type >

Type représente le type de donnée de l'attribut, il en existe trois :
- **littéral** : il permet d'affecter une chaîne de caractères à un attribut, grâce au mot clé CDATA
- **l'énumération** : cela permet de définir une liste de valeurs possibles pour un attribut donné : < ! ATTLIST Elément Attribut (Valeur1 | Valeur2 | ...) > Pour définir une valeur par défaut il suffit de faire suivre l'énumération par la valeur désirée entre guillemets : < ! ATTLIST Elément Attribut (Valeur1 | Valeur2) "valeur par défaut" >
- **atomique** : il permet de définir un identifiant unique pour chaque élément grâce au mot clé ID

Niveau de nécessité de l'attribut :
- #IMPLIED signifie que l'attribut est optionnel
- #REQUIRED signifie que l'attribut est obligatoire
- #FIXED signifie que l'attribut sera affecté d'une valeur par défaut s'il n'est pas défini. Il doit être immédiatement suivi de la valeur entre guillemets

Exemple de DTD

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!ELEMENT etudiant (ou, matiere*)>
<!ATTLIST etudiant
    num ID #REQUIRED
    nom CDATA #REQUIRED
    prenom CDATA #REQUIRED
    annee (ing1|ing2|ing3) "autre" #REQUIRED
    age CDATA #IMPLIED
    >
<!ELEMENT ou (#PCDATA)>
<!ATTLIST ou
   adresse CDATA #REQUIRED
   ville CDATA #REQUIRED
   cp CDATA #REQUIRED
   >
```

2 Définition des fichiers XML par les schémas

Exemple de XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="personne">
   <xs:complexType>
    <xs:sequence>
       <xs:element name="nom" type="xs:string"/>
       <xs:element name="prenom" type="xs:string"/>
       <xs:element name="date_naiss" type="xs:date"/>
    </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

2.1 Element simple

Un élément simple ne peut contenir que du texte typé, et ne peut pas contenir d'attribut
Exemple : <xs :element name="nom" type="xs :string"/>

On peut donner une valeur par défaut : <xs :element name="xxx" type="yyy" default="zzz"/>
On peut fixer une valeur, aucune autre valeur n'est autorisée : <xs :element name="xxx" type="yyy" fixed="zzz"/>

2.2 Restriction de valeurs

On peut définir des restrictions de valeurs dans un élément

```
<xs:element name="age">
 <xs:simpleType>
   <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="120"/>
   </xs:restriction>
 </xs:simpleType>
</xs:element>
```

On peut définir des restrictions d'ensemble de valeurs dans un élément

```
<xs:element name="voiture">
 <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:enumeration value="Picasso"/>
    <xs:enumeration value="Ferrari"/>
   </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Syntaxe permettant de réutiliser la restriction dans d'autres éléments

```
<xs:element name="voiture" type="TypeAuto" />
<xs:simpleType name="TypeAuto">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Picasso"/>
  < xs:enumeration value="Ferrari"/>
 </xs:restriction>
</xs:simpleType
```

On peut imposer une longueur pour un élement simple avec les balises xs :length, xs :minLength et xs :maxLength

```
<xs:element name="password">
  <xs:simpleType>
  <xs:restriction base="xs:string">
     <xs:minLength value="5"/>
     <xs:maxLength value="8"/>
  </xs:restriction>
  </xs:simpleType>
</xs:element>
```

2.3 Element complexe

Un élément complexe est un élément qui contient d'autres éléments et/ou un ou plusieurs attributs

```
<xs:element name="employee">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element
```

Indicateurs de type complexe :
- Indicateurs d'ordre
  - all : tous / ordre indifférent
  - sequence : tous / dans l'ordre
  - choice : un au choix
- Indicateurs d'occurence
  - minOccurs (défaut 1)
  - maxOccurs (défaut 1)
  - valeur infinie : unbounded

Exemple d'indicateur : (elt1,elt2*,(elt3|elt4+))*

```
<xs:sequence minOccurs="0" maxOccurs="unbounded">
 <xs:element name="elt1"/>
 <xs:element name="elt2" minOccurs="0"
             maxOccurs="unbounded"/>
 <xs:choice>
  <xs:element name="elt3"/>
  <xs:element name="elt4" maxOccurs="unbounded"/>
 </xs:choice>
</xs:sequence>
```

Définition d'attributs : <xs :attribute name="xxx" type="yyy"/> Options possibles :
- default="zzz"
- fixed="zzz"
- used="required"/>

3 Introduction aux langages XML et XSL

Un fichier XSL est un fichier XML dont la racine est définie par la balise xsl :stylesheet
Ce fichier définit des règles pour générer une sortie, dont le type est définit par le nœud fils xsl :output

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes"
   encoding="iso-8859-1" />
```

On peut définir des paramètres : <xsl :param name="nomParam">...</xsl :param>
On peut définir des variables locales à un nœud : <xsl :variable name="nomVar">...</xsl :variable>
On peut récupérer le contenu d'une variable ou d'un paramètre : <xsl :value-of select="$nomElement"/> où nomElement est le libellé de la variable ou du paramètre

L'ordre <xsl :value-of select= " ? ? ? " / > envoie dans la sortie le texte qui est contenu dans la zone indiqué par ? ? ?
L'instruction xsl :template permet de définir pour un nœud ce que le processeur XSLT devra faire quand il devra traiter ce nœud
L'instruction xsl :apply-template permet de préciser au processeur XSLT quel patron de noeud il faut appliquer à un endroit précis du fichier XML

La clause select : cette clause est utilisée dans les instructions xsl :apply-template et xsl :for-each et porte sur des nœuds
L'instruction xsl :if permet de préciser au processeur XSLT un traitement conditionnel
<xsl :if test="..." /> ... </xsl :if>

On appelle contexte l'arbre défini par le nœud courant
- La fonction position() renvoie la place du nœud courant dans le contexte de son père
- La fonction count(node-set) renvoie le nombre de nœds de l'ensemble de nœuds node-set passé en argument
- La fonction last() renvoie la taille du contexte du père du nœud courant
- La fonction node() indique tous les nœuds fils du nœud courant

Les opérateurs booléens sont or, and et not(...) Les opérateurs de comparaison =, &lt ; , &gt ; , !=
Exemples

```
<!- on teste si le noeud courant est le dernier fils de son père-->
<xsl:if test="position() = last()">

<!- on teste si la variable v1 est inférieure v2 -->
<xsl:if test="$v1 &lt; $v2">

<!- on teste si le noeud courant à un fils nommé theFils-->
<xsl:if test="theFils">

<!- on teste si le noeud courant à un attribut nommé theAttribut-->
<xsl:if test="@theAttribut">

<!- on teste si le noeud courant à une variable ou un paramètre nommé theParVal
qui vaut val -->
<xsl:if test="$theParVal = 'val' ">
```

L'instruction <xsl :choose> permet de gérer des branchements conditionnels à plusieurs branchements

```
<xls:choose>
  <xsl:when test="...">...</xsl:when>
   ...
   <xsl:otherwise>...</xsl:when>
</xls:choose>
```

4 Notions avancées de XSL

On peut passer un paramètre en ligne de commande comme suit : java -jar saxon.jar xxx.xml yyy.xsl nomPar=valeurPar
Pour récupérer le paramètre dans le fichier xsl, il faut déclarer le paramètre en haut du fichier xsl (paramètre global) comme suit : <xsl :param name="nomPar"/>

Dans les nœuds de définition de variables ou de paramètres, le moteur xslt ne dirige plus les résultats dans la sortie standard mais dans la variable ou dans le paramètre

```
  <xsl:variable name="maVariable">
        <xsl:for-each select="livre">
          <xsl:value-of select="@titre"/>
          <xsl:if test="not(position() = last())">
                  <xsl:text>-</xsl:text>
          </xsl:if>
        </xsl:for-each>
  </xsl:variable>
```

La variable maVariable aura pour contenu la concaténation des titres des livres séparés par des tirets

On peut effectuer des opérations numérique → <xsl :value-of select="2 * @prix"/>

Il existe une autre catégorie de template que le programmeur peut appeler directement
- on utilise le mot-clé **name** à la place de **match**
- ils peuvent recevoir des paramètres comme lors d'un appel de fonction dans un langage procédural

La syntaxe d'appel d'un tel template est la suivante :

```
  <xsl:call-template name="nomTemplate">
       <xsl:with-param name="nomParam1" select="..." />
       ...
       <xsl:with-param name="nomParamn" select="..." />
  </xsl:call-template>
```

Dans le langage XLS, certains symboles ont un sens particulier comme &, <, ... : ce sont les métacaractères
Pour les déspécialiser et pouvoir les écrire, il faut utiliser < ![CDATA[...]]>

4.1 Chemin XPath

Une expression de chemin XPath est une séquence d'étapes
- chemin absolu : commence par /
- chemin relatif : ne commence pas par /

Une étape XPath contient :
- un axe qui spécifie la relation structurale fils, descendants, ancêtres, frères, attributs,...
- un test qui spécifie le type de n½ud (obligatoire)
- un prédicat pour affiner la sélection ⇒ filtrage

XPath propose les axes suivants :

ancestor	Ancestors of the context node
ancestor-or-self	Ancestors, including the context node
attribute	Attributes of the context node (abbreviated "@")
child	Children of the context node (the default axis)
descendant	Descendants of the context node
descendant-or-self	Descendants, including the context node (abbreviated "//")
following	Elements which occur after the context node, in document order
following-sibling	Elements which occur after the context node, in document order and have the same parent as the context node
preceding	Elements which occur before the context node, in document orde (returned in reverse-document order)
preceding-sibling	Elements which occur before the context node, in document orde (returned in reverse-document order) and have the same parent a the context node
namespace	The namespace nodes of the context node
parent	The parent of the context node (abbreviated "..")
self	The context node (abbreviated ".")

4.2 Les prédicats

- number(object) : traduire l'objet sous la forme d'un nombre → renvoie NaN si l'objet ne représente pas un nombre
- sum(noeuds) : renvoie la somme des noeuds après les avoir transformés en nombre
- count(noeuds) : renvoie le nombre de noeuds
- floor(nombre) : arrondi par le bas → floor(3.64) = 3, floor(3.14) = 3
- ceiling(nombre) : arrondi par le haut → ceiling(3.64) = 4, ceiling(3.14) = 4
- round(nombre) : arrondi par le plus proche → round(3.64) = 4, round(3.14) = 3

5 Quelques fonctions classiques

La fonction concat permet de concaténer des chaînes

```
<xsl value-of select="concat($nom,'.gif')"/>
```

La fonction substring a trois paramètres : la chaîne, la position de début et le nombre de caractères à extraire

```
<xsl:value-of select="substring(@noTel,1,2)"/>
```

La fonction substring-before(txt,token) permet d'extraire de la chaîne txt, la sous chaîne située avant la chaîne token
La fonction substring-after(txt,token) permet d'extraire de la chaîne txt, la sous chaîne située après la chaîne token
La fonction string-length(txt) renvoie la longueur de la chaîne txt
La fonction translate(txt,avant,apres) renvoie une copie de la chaîne txt dans laquelle tous les caractères avant ont été remplacés par les caractères apres. Si la chaîne apres est plus courte que la chaîne avant alors des caractères ne seront pas translatés

Charger en mémoire vive un autre document : <xsl :value-of select="document('monFichier.xml')" />

Dans la clause xsl :sort il faut préciser si ce que l'on veut trier est du texte ou du numérique avec l'attribut data-type : <xsl :sort data-type="text"/>, <xsl :sort data-type="number"/>

La fonction format-number reçoit deux paramètres : la valeur et le format pour transformer cette valeur
Dans le format on utilise les caractères 0 et #
- 0 signifie qu'un chiffre non significatif sera remplacé par 0
- # signifie qu'un chiffre non significatif ne sera pas remplacé

format-number(53.51, "000.###")	retourne 053.51
format-number(53.51, "000.000")	retourne 053.510
format-number(53.51, "#.000")	retourne 53.510

Par défaut la virgule est le séparateur de groupe et le point le séparateur de décimal

Commande pour produire un fichier html **java -jar saxon.jar file.xml file.xsl > file.html**