Cours XML

Michel Meynard

14 novembre 2003

1 Introduction

Tout d'abord, un peu de vocabulaire :

XML eXtensible Markup Language ; c'est un dialecte de SGML défini par le World Wide Web Consortium (W3C) ;

SGML Standard Generalized Markup Language ; standard international de l'ISO définissant la structure de différents types de documents électroniques ;

HTML HyperText Markup Language ; le langage du Web ;

SGML est très complexe et admet de nombreuses possibilités et options dans la définition de la syntaxe des documents structurés (manuscrits irlandais, documentation technique, partitions musicales, ...). Au contraire, HTML est un langage simple définissant une classe de documents publiables sur le Web. XML est une version simplifiée de SGML permettant de :

- définir facilement un type de document par une DTD (Document Type Definition) ou un Schéma XML,
- d'écrire rapidement des programmes :
  - vérifiant la syntaxe d'une instance de document par rapport à son type ;
  - traduisant un document dans différents formats (HTML, PDF, RTF, ...).

1.1 Quelques particularités

- XML est sensible à la casse (minuscule, majuscule) ;
- tous les attributs doivent être entourés de guillemets ou d'apostrophes ;
- les instructions de traitement commencent par `<?` et finissent par `?>`.

2 XML, un langage à balises (markup language)

Pour définir un langage à balises, commençons par définir le vocabulaire de base :

balise une balise est soit ouvrante (i.e. `<p>`), soit fermante (i.e. `</p>`) ;

élément un élément XML est composé d'une balise ouvrante, d'un contenu puis d'une balise fermante. Contrairement à HTML, toute balise ouverte doit être fermée : pas de `<p>bla bla.<p>bla bla.`. Un élément vide (sans contenu) peut être défini de 2 façons :
- `<hr></hr>` ;
- `<hr/>`.

entité différents type d'entités existent ; en règle générale, une entité est un nom référençant quelque chose. Cela peut servir pour définir des macros , des abbréviations, des inclusions, ... Entités références `&nom;` et entités paramètres `%nom;` permettent de factoriser les définitions.

attribut un élément peut définir des attributs ayant des valeurs dans sa balise ouvrante sous la forme d'une liste de `nom="valeur"`.

3 Un exemple simple

XML est un métalangage permettant de définir un type de document (DTD) mais aussi une instance de document. Parfois, la DTD est définie dans le même fichier que l'instance, parfois non. Un document XML peut être "DTDless", c'est-à-dire sans définition de la structure du langage comme dans l'exemple 1.

Exemple 1 (rep.xml)

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

```
<entree><nom>Dupont</nom>
<lieu type="domicile"><tel>0123456789</tel></lieu>
<lieu type="bureau"><tel>9876543210</tel></lieu>
</entree>

<entree><nom>Durand</nom>
<lieu type="bureau">
  <tel>0466543210</tel>
  <adrs>12 rue Ada, 34000 Montpellier</adrs>
</lieu>
</entree>
</repertoire>
```

Un outil de validation appliqué à ce fichier fera une simple analyse (parsing) du fichier. C'est-à dire qu'il vérifiera uniquement le bon parenthésage des balises. Un document bien parenthésé est dit **bien formé** (well formed). Si l'on définit une DTD, alors le document sera dit **valide**.

Par exemple, en utilisant la boîte à outils **Jade**, le validateur SGML se nomme `nsgmls`. La ligne de commande doit utiliser le fichier de déclaration `xml.dcl` pour les documents XML : `nsgmls -s c:/Jade/xml.dcl dtdless.xml`

4 Structure de document XML et de DTD

Une instance de document XML est composé de :
- des instructions de traitement `<?...?>`, destinées aux outils de traitement du fichier (validateur, traducteurs) ;
- une définition facultative de type de document (DTD) `<!DOCTYPE ...` ;
- un élément racine ayant le nom du DOCTYPE ;
- un ensemble d'éléments éventuellement attribués et correctement emboîtés ;
- des références aux entités ;

Une DTD est composée de :
- définitions des éléments sous la forme `<!ELEMENT nomElement (definitionElement)>` ; la définition d'un élément peut contenir d'autres éléments, des morceaux de texte `#PCDATA` (Parsed Character DATA), des opérateurs définissant :

  une séquence sous la forme `a, b, c` ;

  une alternative sous la forme `a | b | c` ;

  une répétitive sous la forme `a*` (0..n) ou `a+` (1..n) ;

  une optionelle sous la forme `a?` ;

  des parenthèses pour changer les priorités ;
- définitions des attributs d'un élément sous la forme `<!ATTLIST nomElement listeAttributEtValeur>` ;
- définitions d'entités sous la forme `<!ENTITY mm "Michel Meynard">`.

Chaque attribut peut avoir une valeur libre `CDATA` (Character DATA), ou une valeur parmi plusieurs (énumération avec valeur par défaut), le qualificatif `#REQUIRED` indiquant le caractère obligatoire de l'attribut ...

Le fichier de l'exemple 2 définit une DTD interne et une instance de ce type de document.

Exemple 2 (repDtd.xml)
```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!-- ce qui suit est la DTD -->
<!DOCTYPE repertoire [
  <!ELEMENT repertoire (entree*)>
  <!ELEMENT entree (nom, lieu+)>
  <!ELEMENT nom (#PCDATA)>
  <!ELEMENT lieu (tel, adrs?)>
  <!ELEMENT tel (#PCDATA)>
  <!ELEMENT adrs (#PCDATA)>
  <!ATTLIST lieu
    type CDATA #REQUIRED
  >
]>
<!-- ce qui suit est l'intance -->
<repertoire>

<entree><nom>Dupont</nom>
```

```
<lieu type="domicile"><tel>0123456789</tel></lieu>
<lieu type="bureau"><tel>9876543210</tel></lieu>
</entree>

<entree><nom>Durand</nom>
<lieu type="bureau">
  <tel>0466543210</tel>
  <adrs>12 rue Ada, 34000 Montpellier</adrs>
</lieu>
</entree>

</repertoire>
```

Là encore la validation est réalisée par la commande : nsgmls -s c:/Jade/xml.dcl repertoire.xml

4.1 DTD externe et Identifiants

On peut définir une partie de la DTD en externe et y ajouter des définitions internes avec la syntaxe :

```
<!DOCTYPE racine (SYSTEM "url_de_la_dtd_externe")
              | (PUBLIC "identifiant_public" "url_de_la_dtd_externe"?)
   [ définitions_internes ]
>
```

- SYSTEM est utilisé pour donner l'adresse du fichier qui contient la DTD. Par exemple, <!DOCTYPE repertoire SYSTEM "rep.dtd">, définit le fichier rep.dtd comme DTD du document. Une URI HTTP est également possible.
- PUBLIC est utilisé lorsque la DTD est une norme ou qu'elle est enregistrée sous forme de norme ISO par l'auteur. L'intérêt principal de l'identifiant PUBLIC est de rendre la localisation de la DTD indépendante de la plate-forme locale. Si l'URL n'est pas fournie, la résolution est réalisée par le mécanisme de catalogue (CATALOG). **Attention**, pour l'instant un identifiant système doit être fourni pour XML (soit par SYSTEM, soit par l'url de PUBLIC). L'identifiant public est composé :
  type // proprietaire // DTD description // langue avec :

  type un signe + si l'identifiant est enregistré, un signe - sinon ;

  proprietaire : nom du propriétaire (entreprise ou personne ou ISO) ;

  classe description : classe parmi (DTD, ELEMENTS, ENTITIES) suivie d'une description textuelle ;

  langue : un code de langue ISO 639 (EN, FR, ...).

L'exemple 3 utilise la DTD DocBook XML (existant aussi en SGML) pour illustrer un identifiant PUBLIC :

Exemple 3 (exdocbook.xml)

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook V4.2//EN" "c:/Docbook/docbookx.dtd">
<chapter><title>Premier essai</title>
<para>Hello World !</para>
</chapter>
```

4.2 Entités

Une entité est définie dans la DTD et est ensuite utilisée dans l'instance par une ou plusieurs références. Il y a plusieurs types d'entités :

générales internes : défini par <!ENTITY mm "Michel Meynard"> et utilisé par &mm;. C'est une sorte de macro #define ;

générales externes : défini par <!ENTITY fic1 SYSTEM "fic1.txt"> et utilisé par &fic1;. C'est une sorte d'inclusion #include. Les fichiers binaires (graphiques, ...) doivent être définis par <!ENTITY monimg SYSTEM "dessin.eps" NDATA EPS>. Ils ne peuvent être utilisés que comme valeur d'attribut : <graphic ref="monimg"> </graphic> ;

caractères spéciaux tels que < pour <, & pour &, &#xhhhh; pour les caractères Unicode.

paramètres elles ne sont utilisées que dans les DTD pour les personnaliser. <!ENTITY % brouillon "INCLUDE"> définit %brouillon; comme ayant la valeur INCLUDE. L'utilisation dans l'instance du document sera de la forme :

```
<![%brouillon;[
<para>Visible seulement en mode brouillon.</para>
]]>
```

On pourra au final, modifier la définition du paramètre dans la DTD afin d'annuler l'inclusion :

```
<!ENTITY % brouillon "IGNORE">
```

5 Feuilles de style

L'intérêt principal de XML est l'indépendance de la structure du document par rapport à sa présentation, contrairement à HTML. Pour publier un document XML, il faut utiliser une **feuille de style** et un traducteur. La feuille de style est référencée dans la deuxième ligne du document au format :

```
<?xml-stylesheet type="text/css" href="instrep.css"?>
```

Il existe plusieurs types de feuilles de styles :

CSS les "Cascading Style Sheet" peuvent être utilisées comme en HTML :

```
/* fichier de styles au format CSS */
entree {display: block; color: red}
tel {margin-left:10pt; display: block; color: blue}
adrs {margin-left:50pt; color: green}
```

Les entrées du repertoire seront représentées sous forme arborescente. C'est le navigateur qui traduira le fichier xml, ce qui engendrera des différences d'affichages. Pas utilisé en pratique.

XSL "XML StyLe sheet" décrit des feuilles de styles au format XML ;

DSSL "Document Style Semantics and Specification Language" décrit des feuilles de styles au format XML avec des expressions Scheme pour spécifier la traduction effectuée.

6 XSL

XSL se décompose en 2 parties : XSLT (XSL Transformation) et XSL-FO (Formating Objects). XSLT se charge de transformer (convertir) un document XML en un autre document (le plus souvent XML). XSL-FO se chargera ensuite de la mise en page précise.

7 XSLT

Un document XSLT est une feuille de style qui décrit des transformations à effectuer.

7.1 Processeurs XSLT

Un processeur XSLT prend en entrée un document XML, une feuille de style, et produit un document de sortie résultant des transformations. Ce document de sortie peut être un document XHTML. Dans ce cas, Le processeur peut être lancé depuis la ligne de commande, ou depuis un navigateur (côté client), ou depuis le serveur HTTP (côté serveur). Cette dernière solution est la plus sûre.

Les deux processeurs XSLT les plus courants sont Xalan (Apache) et Saxon.

Saxon L'installation et l'utilisation de Saxon consiste à :

1. télécharger le fichier compressé ;
2. le décompresser dans un répertoire ;
3. positionner `saxon.jar` dans la variable d'environnement `CLASSPATH` ;
4. transformer le document : `java com.icl.saxon.StyleSheet source.xml feuille.xsl`

Xalan Les dernières versions du JSDK intègrent Xalan et on peut donc l'utiliser directement : `java org.apache.xalan.xslt.Process -in source.xml -xsl feuille.xsl -out resultat.txt`. Malheureu-sement, dès que Saxon est dans le `CLASSPATH`, Xalan ne fonctionne plus ! Aussi, si l'on veut utiliser al- ternativement les deux processeurs, il faut spécifier saxon.jar dans l'option classpath de la ligne de commande java : `java -classpath "C :/Saxon/saxon.jar" com.icl.saxon.StyleSheet rep.xml rep.xsl`

Processeur intégré au navigateur Certains navigateurs (Internet Explorer 6.0 et Mozilla 5.0) possèdent un processeur XSLT intégré. Dans ce cas, la liaison à la feuille de style doit être réalisée dans le document xml source :

Internet Explorer 6.0 `<?xml-stylesheet type="text/xsl" href="./feuille.xsl"?>`

Mozilla 5.0 `<?xml-stylesheet type="application/xsl" href="./feuille.xsl"?>`

Attention, les deux instructions de traitement diffèrent sur leur type et ne sont pas compatibles ! Conclusion, il ne faut pas faire confiance au navigateur pour le traitement XSLT.

7.2 Feuille de style XSLT

Une feuille de style XSLT est un document XML dont la racine est un élément : <xsl :stylesheet version="1.0" xmlns :xsl= "http ://www.w3.org/1999/XSL/Transform">. Les espaces de nom (XML Name Space) permettent d'utiliser le même nom d'élément dans différents contextes sans qu'il y ait ambiguité. Par exemple, xsl :variable est un élément différent de toto :variable. Dans l'exemple 4, une feuille de style minimale est fournie :

Exemple 4 (mini.xsl)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Appliquée au document de l'exemple 1, elle produira le fichier texte suivant :

Exemple 5 (min.txt)

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

Dupont
0123456789
9876543210


Durand

  0466543210
  12 rue Ada, 34000 Montpellier
```

Cette transformation a supprimé toutes les balises en ne conservant que les caractères de texte et les blancs. Par défaut, les noeuds XML non traités sont ainsi reproduits textuellement dans le résultat.

7.3 Le langage XSLT

La transformation est effectuée depuis l'arbre XML source vers un arbre résultat, puis l'arbre destination est sérialisé en un fichier texte. **Attention, la construction de l'arbre résultat impose que le texte littéral inséré dans la feuille de style soit bien formé** ! Par exemple, l'insertion de texte mal balisé est interdit. Le langage est composé de règles de transformations dont la principale est xsl :template, qu'on nommera **modèle** en Français. Celle-ci permet de décrire la transformation à effectuer lorsqu'on rencontre un noeud de l'arbre XML qui corrrespond (match) avec le modèle. Dans l'exemple 6, une règle template est utilisée pour sélectionner chaque noeud <entree> et le transformer en la valeur <xsl :value-of ...> de l'élément <nom> suivi d'un passage à la ligne.

Exemple 6 (rep1.xsl)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="entree">
  <xsl:value-of select="nom"/><br/>
</xsl:template>
</xsl:stylesheet>
```

Appliquée au document de l'exemple 1, le résultat produit par cette feuille de style est :

Exemple 7 (rep1.txt)

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

Dupont<br/>

Durand<br/>
```

Attention, si l'on ne gère pas la racine ou ses fils directs, il ne faut pas oublier que la règle par défaut consiste à reproduire les noeuds de texte sous-jacents à ces derniers. C'est pourquoi, une bonne programmation XSLT consiste à toujours définir un modèle pour la racine (/) ou pour ses enfants directs.

7.4 Récursivité

Il y a deux styles de programmation en XSLT : l'itérative et la récursive. Bien entendu, comme on travaille sur un arbre qui est une structure récursive, cette dernière approche est la plus adéquate. Pour appliquer récursivement des modèles aux enfants d'un élément, l'élément `<xsl :apply-templates/>` permet d'appeler récursivement tous les autres modèles définis dans la feuille de style. On peut limiter cet élément en spécifiant l'attribut `select` correspondant au noeuds sous-jacents que l'on veut traiter.

L'exemple 8 illustre cet élément :

Exemple 8 (rep2.xsl)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">   <!-- racine -->
 <html><head></head>       <!-- en-tête HTML -->
 <body><H1>Répertoire</H1><table>
  <xsl:apply-templates/>   <!-- appliquer les autres modèles -->
 </table></body></html>
</xsl:template>

<xsl:template match="entree">
  <tr><td><b><xsl:value-of select="nom"/></b></td></tr>
  <xsl:apply-templates select="lieu"/>   <!-- appliquer le modèle de lieu -->
</xsl:template>

<xsl:template match="lieu">
  <tr><td><xsl:value-of select="@type"/></td> <!-- l'@ttribut type -->
  <td><xsl:value-of select="adrs"/></td>
  <td><xsl:value-of select="tel"/></td></tr>
</xsl:template>

</xsl:stylesheet>
```

Appliquée au document de l'exemple 1, le résultat produit par cette feuille de style est :

Exemple 9 (rep2.html)

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
   </head>
   <body>
      <H1>R&eacute;pertoire</H1>
      <table>


         <tr>
            <td><b>Dupont</b></td>
         </tr>
         <tr>
            <td>domicile</td>
            <td></td>
            <td>0123456789</td>
         </tr>
         <tr>
            <td>bureau</td>
            <td></td>
            <td>9876543210</td>
         </tr>


         <tr>
            <td><b>Durand</b></td>
         </tr>
         <tr>
```

```
            <td>bureau</td>
            <td>12 rue Ada, 34000 Montpellier</td>
            <td>0466543210</td>
         </tr>

      </table>
   </body>
</html>
```

Remarquons la désignation de la valeur d'un attribut par le préfixe "@". Remarquons également que l'en-tête HTML a remplacé l'en-tête XML, car le processeur XSLT s'est aperçu de la nature de la racine de l'arbre produit !

8 XPath

XPath (XML Path language) est un langage sans balise permettant de spécifier un ou plusieurs éléments (noeuds) d'un document XML. Il est basé sur la notation Unix des fichiers et répertoires. Les expressions XPath correspondent (match) à différents noeuds selon le contexte dans lequel ces expressions sont utilisées. L'attribut `match` des modèles XSLT ainsi que l'attribut `select` des éléments `xsl :value-of` et `xsl :apply-templates` sont des expressions XPath. Les exemples suivants illustrent le langage XPath :

/ la racine de l'arbre XML ;

`*` tous les noeuds ;

. le noeud courant ;

.. le parent du noeud courant ;

entree/nom les `<nom>` enfants d'une `<entree>` ;

//tel les éléments `<tel>` où qu'ils soient dans l'arbre ;

repertoire//adrs les noeuds `<adrs>` descendants de `<repertoire>` ;

../entree[1] le premier noeud `<entree>` enfant du parent du noeud courant ;

lieu[@type="bureau" |lieu[adrs]] les noeuds `<lieu>` ayant un attribut de `type` valant bureau ainsi que les noeuds `<lieu>` ayant un enfant `<adrs>` ;

Voici un exemple d'utilisation des expressions XPath dans différents éléments de XSLT :

Exemple 10 (rep3.xsl)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">   <!-- racine -->
 <html><head></head>       <!-- en-tête HTML -->
 <body><H1>Répertoire</H1><table>
  <xsl:apply-templates select="//lieu"/> <!-- seulement les modèles de lieu -->
 </table></body></html>
</xsl:template>

<xsl:template match="lieu"/> <!-- règle par défaut : rien -->
<xsl:template match="lieu[@type='domicile']|lieu[adrs]"> <!-- plus spécifique-->
  <tr><td><xsl:value-of select="../nom"/></td> <!-- nom du père -->
  <td><xsl:value-of select="adrs"/></td>
  <td><xsl:value-of select="tel"/></td></tr>
</xsl:template>

</xsl:stylesheet>
```

Appliquée au document de l'exemple 1, le résultat produit par cette feuille de style est :

Exemple 11 (rep3.html)

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
   </head>
   <body>
      <H1>R&eacute;pertoire</H1>
```

```
        <table>
            <tr>
                <td>Dupont</td>
                <td></td>
                <td>0123456789</td>
            </tr>
            <tr>
                <td>Durand</td>
                <td>12 rue Ada, 34000 Montpellier</td>
                <td>0466543210</td>
            </tr>
        </table>
    </body>
</html>
```

Outre l'utilisation d'expressions XPath, l'exemple 10 illustre la gestion des conflits de modèle. Il existe deux modèles applicables pour les noeuds de lieu qui sont de type domicile ou qui possèdent une adresse. Le modèle le plus spécifique est celui qui sera utilisé par le processeur XSLT. Cela provient de règles de priorité associé à chaque construction de XPath. Remarquons que le premier modèle de lieu ne produit rien, ce qui évite la production du texte qui serait effectué par les modèles implicites. Ces derniers sont décrit dans l'exemple 12 suivant :

Exemple 12 (implicite.xsl)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/|*">   <!-- racine et tout autre noeud -->
  <xsl:apply-templates/> <!-- descendre récursivement -->
</xsl:template>
<xsl:template match="text()"> <!-- par défaut pour les noeuds contenant du texte -->
  <xsl:value-of select="."/>
</xsl:template>
</xsl:stylesheet>
```

Appliqué à tout fichier XML, cette feuille de style a le même effet que la feuille `mini.xsl` de l'exemple 4.

8.1 Syntaxe XPath

Il existe 7 types de noeuds dans un arbre XML :

- la racine qui possède au moins 1 noeud enfant : l'élément racine du document XML (`repertoire` dans l'exemple 1) ;
- les éléments comme `lieu` ou `tel` ;
- les noeuds de texte comme `Dupont` ou `0123456789` ;
- les noeuds d'attribut comme `type="bureau"` ;
- les noeuds de commentaire ;
- les instuctions de traitement ;
- les espaces de nom ;

Les expressions XPath peuvent être abrégées comme dans les exemples précédents, ou utiliser une syntaxe plus verbeuse mais plus complète décrite ci-après. Plusieurs expressions peuvent être combinés avec l'opérateur | pour exprimer une union. Une expression XPath est constituée d'une suite de mots (dits étapes de localisation) séparés par des /. Chaque mot est constitué :

- d'un axe optionnel suivi de `::` ;
- d'un test de noeud obligatoire ;
- d'un prédicat optionnel encadré par des crochets [].

8.1.1 Axe de recherche

Un axe spécifie une direction de recherche par rapport au noeud courant :

ancestor remontée dans l'arbre jusqu'à la racine ;

descendant descente sur tous les descendants ;

following tous les noeuds suivants dans l'ordre du document XML ;

preceding tous les noeuds précédents ;

following-sibling tous les noeuds suivants ayant le même parent ;

self, parent, attribute, descendant-or-self abrégés en ., .., @, //.

8.1.2 Test de noeud

Un test de noeud est le plus souvent un nom d'élément mais il peut aussi être :

`*` caractère joker correspondant à tous les noeuds éléments (ni noeud attribut, ni noeud texte, ...) ;

@* tous les noeuds attributs ;

node() tout noeud ;

text() tout noeud texte ;

comment() tout noeud de commentaire ;

processing-instruction() tout noeud de traitement ;

8.1.3 Prédicat

Un ou plusieurs prédicats entre crochet peut suivre le test de noeud afin de spécialiser encore la recherche. XPath reconnaît 4 types de données : ensemble de noeuds, booléen, nombre, chaîne. Le prédicat peut utiliser :

des opérateurs de comparaisons =, !=, <, >,<=, >= ;

des fonctions booléennes true(), false() ;

des opérateurs booléens and, or, not ;

des opérateurs numériques +, -, *, div, mod ;

des fonctions de chaînes concat(s1,s2), contains(contenant,contenu), starts-with(s1,debut), substring(s,debut,longueur) (chaînes indicées à partir de 1) ;

des fonctions sur les ensembles de noeuds count(ens), last(), position(). Chaque noeud a une position indicée à partir de 1 par rapport à son parent. Ainsi, lieu[1] référence le premier enfant de type lieu du noeud courant ;

Quelques exemples d'expressions XPath complexes suivent :

//entree[position()=1]/lieu[@type='bureau'][not(adrs)]/tel[starts-with(.,'98')] ; correspond à l'ensemble réduit au deuxième numéro de téléphone de l'exemple 1 ; remarquons que le test [position()=n] est souvent abrégé en [n] ;

substring(.,1,4) les quatres premières lettres du noeud courant ;

//entree/lieu[position()=last()] correspond aux derniers lieu de chaque entree ;

9 Autres éléments XSLT

9.1 Modes

Afin de pouvoir traiter le même noeud plusieurs fois de différentes manières, on peut ajouter un attribut mode aux éléments xsl :apply-templates et xsl :template. Un modèle ayant un mode ne sera appliqué que lorsque un xsl :apply-templates correspondant sera activé. L'exemple 15 illustre l'utilisation de mode pour générer un index hypertexte des noms du répertoire.

9.2 Création d'éléments et d'attributs dans l'arbre résultat

L'arbre résultant de la transformation doit être bien formé. Aussi, on peut insérer litéralement des éléments XHTML comme dans l'exemple 8 ou 10 avec les balises table, tr, td. Par contre, si l'on veut créer des éléments variables, il faut utiliser l'élément XSLT <xsl :element name='elt'> qui contiendra éventuellement des éléments XSLT <xsl :attribute name='att'> et du texte ou des éléments litéraux. L'exemple 15 illustre la création d'ancres XHTML sources et cibles.

9.3 Tri

L'élément XSLT <xsl :sort select='exp' ...> permet de trier un ensemble de noeuds suivant l'expression de tri exp. Ce tri ne peut être effectué qu'en tant qu'enfant de xsl :apply-templates ou de xsl :for-each. L'exemple 15 illustre le tri dans l'ordre inverse des noms du répertoire.

9.4 Alternatives

L'élément XSLT <xsl :if test='exp-bool'> permet d'effectuer une alternative simple (sans sinon). Un tel test est présent dans l'exemple 13. L'élément XSLT <xsl :choose> permet d'effectuer une alternative multiple (switch) à l'aide des éléments <xsl :when test='exp-bool'> et de l'élément <xsl :otherwise>. L'exemple 15 illustre une telle alternative.

9.5 Procédures et paramètres

On peut utiliser des modèles (`xsl :template`) nommés (attribut `name` et pas d'attribut `match`) pour simuler des procédures. Dans ce cas, le noeud courant reste le même dans l'appelé que l'appelant. Le passage de paramètres est possible avec les éléments `xsl :with-param` dans l'appelant avec un attribut `name` et un attribut `select` pour le couple nom, valeur. Dans l'appelé, on introduira juste après le modèle nommé un élément `xsl :param` avec un attribut `name` identique et une valeur par défaut `select` optionnelle. Pour obtenir la valeur d'un paramètre (le déréférencer), il faut le préfixer par le symbole $. Attention, dans une expression XPath, il ne faut pas hésiter à l'entourer par des parenthèses : `($x)-1`.

L'exemple 13 illustre l'utilisation d'une procédure récursive `hanoi` qui affiche les déplacements de disques à réaliser pour transférer n disques de la tour 1 vers la tour 3 en utilisant une tour intermédiaire 2, sans déplacer plus d'un disque à chaque fois et sans poser un disque de diamètre supérieur sur un plus petit. La valeur de n est le nombre d'éléments entree du document XML.

Exemple 13 (hanoi.xsl)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/> <!-- ou bien html, ou bien text -->
<xsl:template match="/">   <!-- racine -->
 <html><head></head><body>       <!-- en-tête HTML -->
  <H1><xsl:text>Les tours de Hanoi avec </xsl:text>
   <xsl:value-of select="count(//entree)"/><xsl:text> disques</xsl:text>
  </H1>
  <xsl:call-template name="hanoi">
    <xsl:with-param name="num" select="count(//entree)"/> <!-- nb de disques -->
    <xsl:with-param name="depart" select="1"/> <!-- tour de départ -->
    <xsl:with-param name="arrivee" select="3"/> <!-- tour d'arrivée -->
  </xsl:call-template>
 </body></html>
</xsl:template>

<xsl:template name="hanoi">
  <xsl:param name="num"/> <!-- numéro du disque -->
  <xsl:param name="depart"/> <!-- tour de départ -->
  <xsl:param name="arrivee"/> <!-- tour d'arrivée -->
  <xsl:if test="$num!=0">
    <xsl:call-template name="hanoi"> <!-- 1er appel récursif -->
    <xsl:with-param name="num" select="($num)-1"/> <!-- 1 disque de moins -->
    <xsl:with-param name="depart" select="$depart"/> <!-- même tour de départ -->
    <xsl:with-param name="arrivee" select="6-($depart)-($arrivee)"/> <!-- tour intermédiaire -->
    </xsl:call-template>
    <xsl:text>
    Déplacement du disque </xsl:text><xsl:value-of select="$num"/>
    <xsl:text> depuis la tour </xsl:text><xsl:value-of select="$depart"/>
    <xsl:text> vers la tour </xsl:text><xsl:value-of select="$arrivee"/>
    <br/>
    <xsl:call-template name="hanoi"> <!-- 2ème appel récursif -->
    <xsl:with-param name="num" select="($num)-1"/> <!-- 1 disque de moins -->
    <xsl:with-param name="depart" select="6-($depart)-($arrivee)"/> <!-- tour intermédiaire -->
    <xsl:with-param name="arrivee" select="$arrivee"/> <!-- même tour d'arrivée -->
    </xsl:call-template>
  </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Appliquée au document de l'exemple 1, le résultat produit par cette feuille de style est :

Exemple 14 (hanoi.html)

```
<?xml version="1.0" encoding="utf-8"?><html><head/><body><H1>Les tours de Hanoi avec 2 disques</H1>
    DÃľplacement du disque 1 depuis la tour 1 vers la tour 2<br/>
    DÃľplacement du disque 2 depuis la tour 1 vers la tour 3<br/>
    DÃľplacement du disque 1 depuis la tour 2 vers la tour 3<br/></body></html>
```

9.6 Autres éléments utiles

– L'élément `<xsl :text>` permet d'insérer un noeud texte dans l'arbre résultat. Sa seule utilité par rapport à du texte littéral est de permettre de formater sur plusieurs lignes le texte sans que ces sauts de lignes se retrouvent dans le résultat ;
– les paramètres peuvent également être utilisés dans un `xsl :apply-templates` mais malheureusement l'évaluation du paramètre n'est fait qu'une seule fois avant le premier "appel" et sa valeur reste donc constante (peu utile) ! De la même facon, l'élément `xsl :variable` permet d'utiliser des constantes nommées dont la valeur est calculée une seule fois ! De plus, ces constantes ne sont pas réaffectables !
– L'élément `<xsl :for-each select="elt">` permet d'itérer sur l'ensemble des éléments sélectionnés. Ces éléments peuvent être triés par un `xsl :sort`. Cet élément est utilisé dans l'exemple 15 ;
– l'élément `xsl :output method="html"` permet de spécifier une méthode de sérialisation de l'arbre résultat parmi `xml, html, text`. Par défaut, c'est la méthode `xml` qui est utilisée sauf si le processeur XSLT sait reconnaître une racine `<HTML>`.
– l'élément `xsl :message` permet de générer des messages d'erreurs à des fins de débogage ; la ligne où est situé le message est indiqué.

Tous ces éléments se retouvent dans l'exemple 15 qui consiste à réaliser un répertoire téléphonique composé d'un index des noms des personnes, et du répertoire constitué du nom et des téléphones associés aux lieux. La difficulté, pricipale de cet exemple est la génération des ancres liées à la position de chaque entrée alors que l'on est sur un noeud de lieu !

Exemple 15 (rep7.xsl)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/">   <!-- racine -->
 <html><head></head><body>       <!-- en-tête HTML -->
  <H1>Index</H1>
  <xsl:apply-templates select="//entree" mode="index">
    <xsl:sort select="nom" order="descending"/> <!-- tri sur les noms -->
  </xsl:apply-templates>
  <H1>Répertoire téléphonique</H1><table>
  <xsl:apply-templates select="//entree">
    <xsl:sort select="nom" order="descending"/> <!-- tri sur les noms -->
  </xsl:apply-templates>
  </table></body></html>
</xsl:template>

<xsl:template match="entree" mode="index"> <!-- INDEX des noms -->
  <xsl:element name="a"> <!-- création d'un lien html -->
    <xsl:attribute name="href">
      <xsl:value-of select="concat('#',position())"/>
    </xsl:attribute>
    <xsl:value-of select="nom"/>
  </xsl:element>;
</xsl:template>

<xsl:template match="entree"> <!-- Liste des noms et téléphones -->
 <xsl:variable name="pos" select="position()"/>
 <xsl:for-each select="lieu">
  <xsl:sort select="@type"/>
  <xsl:call-template name="lieu">
   <xsl:with-param name="num-entree">
    <xsl:value-of select="$pos"/>
   </xsl:with-param>
  </xsl:call-template>
 </xsl:for-each>
</xsl:template>

<xsl:template name="lieu"> <!-- procédure appelée pour chaque entree/lieu -->
  <xsl:param name="num-entree"/> <!-- position de l'entree -->
  <tr><td>
```

```
  <xsl:choose>
    <xsl:when test="position()=1"> <!-- Si c'est le premier lieu -->
      <xsl:element name="a"> <!-- création d'une ancre -->
        <xsl:attribute name="name">
          <xsl:value-of select="$num-entree"/>
        </xsl:attribute>
        <xsl:value-of select="../nom"/><!-- contenu de l'ancre -->
      </xsl:element>
    </xsl:when>
    <xsl:otherwise/> <!-- dans les autre cas : rien -->
  </xsl:choose>
  </td>
  <td><xsl:value-of select="@type"/></td>
  <td><xsl:value-of select="tel"/></td></tr>
  <xsl:if test="position()=last()">
    <tr><td colspan="3"><hr/></td></tr> <!-- un filet entre entree -->
  </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Appliquée au document de l'exemple 1, le résultat produit par cette feuille de style est :

Exemple 16 (rep7.html)

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
   </head>
   <body>
      <H1>Index</H1><a href="#1">Durand</a>;
      <a href="#2">Dupont</a>;

      <H1>R&eacute;pertoire t&eacute;l&eacute;phonique</H1>
      <table>
         <tr>
            <td><a name="1">Durand</a></td>
            <td>bureau</td>
            <td>0466543210</td>
         </tr>
         <tr>
            <td colspan="3">
               <hr>
            </td>
         </tr>
         <tr>
            <td><a name="2">Dupont</a></td>
            <td>bureau</td>
            <td>9876543210</td>
         </tr>
         <tr>
            <td></td>
            <td>domicile</td>
            <td>0123456789</td>
         </tr>
         <tr>
            <td colspan="3">
               <hr>
            </td>
         </tr>
      </table>
   </body>
</html>
```

9.7 Conclusion sur XSLT

XSLT est un langage de programmation un peu déroutant qui permet de transformer un arbre XML source en un arbre résultat (XML, HTML, texte) puis de le sérialiser en un flot de caractères. C'est un langage fonctionnel (à la Lisp), par conséquent, il ne faut pas recourir à des variables pour conserver l'information. On utilisera plutôt des procédures (template nommé) avec des paramètres. Les structures de contrôle alternatives (`if, choose`) et répétitives (`for-each`) permettent une programmation traditionnelle.

9.7.1 Erreurs courantes

- oubli de fermeture d'un élément vide (sans contenu) : `<.../>` ;
- mauvaise orthographe d'un élément XSLT : `<xsl :foreach ...>` (il manque le trait d'union) ;
- erreur de correspondance dans le motif : ne pas oublier que x/y fait références aux éléments **y** enfants de x (et pas aux éléments x) ;
- si on installe 2 processeurs XSLT java, tels que Saxon et Xalan, le `CLASSPATH` contient des archives jar incompatibles ! Il y a conflit sur la classe `XSLTErrorResources_fr`.

10 Formating Objects (XSL-FO)

`XSL-FO` est une application XML dédiée à la publication **papier** de documents XML. Généralement, le document XML source sera transformé via une feuille de style XSLT en un document XSL-FO par un processeur XSLT. Puis, un processeur XSL-FO traduira ce dernier en un fichier `pdf`.

10.1 Processeur FO (Formateur) FOP

Un processeur XSL-FO prend en entrée un document XSL-FO d'extension `.fo`, et produit un document de sortie d'extension `.pdf` ou `.ps` ou ... Il peut également produire une sortie graphique sur écran (`awt`). Le processeur peut être lancé depuis la ligne de commande, ou depuis une application (serveur HTTP, ...).

Le formateur le plus courant est FOP (Apache) qui est un programme Java. On l'utilise à l'aide d'un script shell ou batch selon le système d'exploitation. Pour formater le fichier `mini.fo` en `mini.pdf`, il faudra faire : `fop.sh mini.fo -pdf mini.pdf`. Pour formater le fichier `mini.fo` en `mini.ps`, il faudra faire : `fop.sh mini.fo -ps mini.ps`.

On peut également combiner la transformation XSLT et le formatage en utilisant la commande : `fop.sh rep.xml -xsl rep.xsl -pdf rep.pdf`.

10.2 Structure d'un document XSL-FO

C'est un document XML contenant des éléments XSL-FO dans 2 parties distinctes :

1. la première partie décrit les modèles de mise en page du document (`layout-master-set`). Il peut y avoir plusieurs modèles (`master`) dans des documents complexes possédant différentes tailles de pages, différentes régions de texte (en-tête, pied de page, ...). Dans l'exemple 17, un unique modèle nommé toto décrit une mise en page simplissime avec un format 15cmx4cm et des marges de 1cm autour d'une unique région centrale (`region-body`) ;
2. la seconde partie est constituée d'un ensemble de séquences de pages (`page-sequence`) dont chacune est associée à un modèle. Dans chaque séquence de page, des flots (`flow`) de blocs (`block`) de texte (paragraphes) sont associés aux différentes régions définies dans leur modèle. Dans l'exemple 17, un unique flot est inséré dans la région centrale ;

Dans l'exemple 17, un document XSL-FO minimal est donné. Dans la figure 1, le document pdf produit par fop est fourni.

Exemple 17 (mini.fo)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"> <!-- racine XML -->

<fo:layout-master-set> <!-- modèles de mise en page (1 seul modèle "toto") -->
<fo:simple-page-master master-name="toto" page-width="15cm" page-height="5cm"
                       margin="1cm">
  <fo:region-body/> <!-- seule région obligatoirement définie -->
</fo:simple-page-master>
</fo:layout-master-set>

<!-- contenu du document -->
<fo:page-sequence master-reference="toto"> <!-- référence à TOTO -->
```

```
<fo:flow flow-name="xsl-region-body"> <!-- référence à la région -->
<fo:block> <!-- 1er paragraphe -->
Ici, on peut commencer à décrire le texte qui sera inséré dans le corps (body)
de la page ayant comme modèle <fo:inline font-weight="bold">toto</fo:inline>.
</fo:block>
<fo:block> <!-- 2ème paragraphe -->
Et voici la suite du texte ...
</fo:block>
</fo:flow>
</fo:page-sequence>
</fo:root>
```

Ici, on peut commencer à décrire le texte qui sera inséré dans le
corps (body) de la page ayant comme modèle **toto**.
Et voici la suite du texte ...

FIG. 1 – **mini.pdf**

11 DSSL

Les feuilles de styles DSSL ont été conçues pour les documents SGML, et notamment pour l'application DocBook. Une feuille de style DSSL permet de décrire le traitement "à la lisp" à appliquer à chaque élément. Par exemple, dans l'exemple 18, l'élément `nom` sera formatté comme un paragraphe espacé de 10 points par rapport au précédent.

Exemple 18 (mini.dssl)

```
(element nom
  (make paragraph
    space-before: 10pt
    (process-children)))
```

Pour appliquer une feuille à notre document XML de l'exemple 1, il faut utiliser un processeur DSSL tel que `Jade`. Pour cela, il faut recopier dans le répertoire courant les fichiers de Jade suivants :

catalog fichier texte qui décrit l'association entre identifiant PUBLIC de DTD et identifiant système ;

dsssl.dtd la DTD de DSSSL ;

style-sheet.dtd la dtd de style-sheet ;

fot.dtd la dtd du langage XML FOT (Flow Object Tree) ;

Dans l'exemple 19, une feuille de style DSSSL est donnée pour le document de l'exemple 1.

Exemple 19 (repDtd.dsl)

```
<!doctype style-sheet PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">

(root (make simple-page-sequence
(make scroll))) ;page
(element entree  ;nouveau paragraphe espacé
  (make paragraph
    start-indent: 1cm ;1cm du bord
    space-before: 20pt
    (process-children) ;traiter récursivement
    ))
(element lieu ;nouvelle ligne
  (make paragraph
    start-indent: 1.5cm ;du bord
    (process-children)))
(element tel
  (make line-field ;nouveau champ de texte (inline)
   (literal "Tél: ")
   (process-children)))
```

Puis, il ne reste plus qu'à exécuter la transformation en `rtf` :

```
c:/Jade/jade -t rtf -d repDtd.dsl -o rep.rtf c:/Jade/xml.dcl repDtd.xml
```

Enfin on peut voir le résultat dans le fichier `rep.rtf` :

```
Dupont
   Tél: 0123456789
   Tél: 9876543210
Durand
   Tél: 0466543210  12 rue Ada, 34000 Montpellier
```

12 Glossaire

Suivent les définitions des principâles notions XML :

catalog (catalogue) fichier associant chaque identifiant public (PUBLIC) à un identifiant système (SYSTEM) ; chaque ligne du catalogue est de la forme : `PUBLIC "idpublic" "idsystème"` ;

PUBLIC