Cahiers GUTenberg

XML ET XSL : UN NOUVEAU DÉPART POUR LE WEB

Michel Goossens

Cahiers GUTenberg, nº 33-34 (1999), p. 3-126.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_1999___33-34_3_0>

XML et XSL : un nouveau départ pour le web

Michel GOOSSENS

CERN, Division IT, CH-1211 Genève 23, Suisse, <Michel.Goossens@cern.ch>

Résumé. Fin 1996, le W3C et les plus grandes entreprises développant des logiciels pour l'internet se sont mis d'accord de collaborer pour définir un langage de balisage normalisé et optimisé pour le web : ce fut la naissance de XML (*eXtensible Markup Language* ou « langage extensible de balisage »). XML est basé sur SGML, dont il élimine un grand nombre de fonctions peu utilisées ou jugées trop complexes. De plus il évite la plupart des limitations inhérentes à HTML. Après une introduction au langage XML, nous décrivons XSL (*eXtensible Style Language*) qui permet la présentation et la transformation de l'information XML. Pour terminer nous disons quelques mots sur d'autres développements récents du monde XML.

Abstract. *Late in 1996, the W3C and several major software vendors decided to define a markup language specifically optimized for the web:* XML *(eXtensible Markup Language) was born. It is a simple dialect of* SGML*, which does not use many of* SGML*'s seldom-used and complex functions, and does away with most limitations of* HTML*. After an introduction to the* XML *standard, we describe* XSL *(eXtensible Stylesheet Language) for presenting and transforming* XML *information. Finally we say a few words about other recent developments in the* XML *arena.*

Mot-clés : SGML, XML, XSL, HTML, XPath

1. Vers une description structurelle des documents

Dans l'environnement informatisé actuel, il est important que nos documents soient mis à la disposition d'une communauté aussi large que possible. Pour optimiser les possibilités de réutilisation de ces documents par les différents supports, visualisateurs, bases de données, etc., il est primordial que les documents soient clairement balisés structurellement, portables et indexables pour faciliter les recherches. Ceci nécessite une normalisation maximale à tous les niveaux. De plus il faut que les différents acteurs du monde de l'informatique suivent ces normes, ce qui n'est jamais acquis *a priori*. C'est pourquoi un maximum de développeurs doivent être impliqués dans la phase de définition de nouvelles normes, ce qui fut le cas de XML, dont nous parlerons plus loin.

1.1. Au début était SGML

Pour normaliser la description de la structure d'un document, SGML fut adopté par l'ISO en 1986[38]. Toutefois, SGML, qui propose à la fois une approche générale et ouverte, est assez complexe à mettre en œuvre. Pour la grande majorité des documents, surtout ceux distribués sur l'internet, 20% seulement de la fonctionnalité de SGML est utilisée. À l'origine, HTML tentait également de séparer les aspects *structure* et *présentation* d'un document. Malheureusement, Netscape et Microsoft, qui se livrent une bataille féroce pour contrôler le marché juteux de la distribution de l'information sur l'internet, en particulier grâce à leurs butineurs respectifs, recherchent principalement des gains commerciaux immédiats. Ainsi, pour attirer les utilisateurs, ils ont pollué le langage HTML par des extensions « maison », qui mettent en évidence les performances spécifiques de leurs produits. Évidemment ces extensions ne sont pas compatibles avec la spécification HTML comme définie par le W3C (*World-Wide Web Consortium*), ce qui rend le gestion des documents extrêmement problématique.

1.2. Puis vint XML

En réaction à cette tour de Babel des dialectes HTML, plusieurs acteurs du monde internet, comme Microsoft, Netscape et Sun, ont travaillé ensemble et ont défini un nouveau langage, qui combine les avantages de SGML et HTML : le langage de balisage extensible XML (*Extensible Markup language*) qui fut introduit officiellement le 10 février 1998 avec la publication de la recommandation XML[46]. XML est une version simplifiée de SGML qui doit assurer le libre échange des documents entre différentes applications internet.

2. Le langage XML

Tout comme son parent SGML [16], XML[24] est un *méta-langage*. Il permet la définition d'un langage pour décrire la structure de l'information pour un type de document donné en proposant un formalisme normalisé pour construire un langage de balisage adapté aux besoins d'une application spécifique. Ainsi XML et SGML diffèrent essentiellement de HTML, qui est une *application* de SGML avec une syntaxe figée et non extensible.

Pour chaque classe de documents XML (par exemple la famille HTML) nous pouvons définir une DTD (*Data Type Declaration* ou déclaration de classe de document), qui décrit les éléments et leurs attributs, les relations, le contenu, etc.

Pour simplifier le traitement des documents sur le web, XML n'impose pas la présence d'une DTD. En fait, un document XML doit avant tout être « *bien-formé* » : il se compose d'un ou plusieurs éléments, dont un élément *racine*, qui contient tous les autres.

De plus tous les éléments du document, délimités pas leurs balises de début et de fin, doivent être imbriqués correctement.

Un document XML bien-formé qui est correct par rapport à une DTD est dit « valide ». Dans ce cas on vérifie le document par rapport à sa grammaire.

Un document XML peut contenir des données textuelles ou binaires. Pour les *données binaires* aucune contrainte n'est imposée ; il faut seulement qu'elles soient associées à une *notation*, identifiée par un nom, qui doit être communiqué aux applications traitant les données en question. Un fichier XML de type *textuel* contient des séquences de *caractères*. Un caractère est l'unité atomique d'un texte ; tout caractère spécifié dans la norme Unicode[35, 2, 1] est autorisé.

Un *nom* XML commence par une *lettre* (caractère alphabétique ou idéographique d'Unicode) ou le caractère de soulignement (_) qui peut être suivi par une suite de caractères Unicode (à l'exception de quelques-uns).

Un texte XML est un mélange de données de type caractères et de *marques*. Ce marquage consiste en balises de début et balises de fin, d'éléments vides, de références de type entité ou caractère, des commentaires, des DTD et des instructions de traitement. Pour des raisons de place nous nous limiterons aux plus importantes.

2.1. Un document XML simplissime

Le document XML le plus simple (et « bien-formé ») a la forme :

```
<a>Je suis un document XML.</a>
```

On n'a nul besoin d'une DTD et c'est l'application XML (p. ex. le programme de visualisation) qui devra interpréter les actions associées à la balise `<a>`. Un autre fichier un peu plus complet, mais toujours pas valide, comme nous ne spécifions pas de DTD associé, est le suivant :

```
<?xml version="1.0"?>
<poème>Clair de Lune</poème>
```

Notons l'emploi d'un caractère accentué dans le nom de l'élément `poème`. En effet, XML permet l'utilisation de noms d'éléments, d'attributs, etc. en russe, arabe, chinois ou japonais, ce qui n'a pas manqué d'accélérer le niveau d'adoption de la nouvelle norme dans les pays non-anglophones.

Le système de balisage est construit à partir de délimiteurs, de symboles spéciaux, de mots-clé ayant une signification particulière. Par exemple si « `para` » identifie l'élément « paragraphe », `<para>` sera la balise qui, dans le document, marque le début d'un paragraphe, « < » et « > » étant les délimiteurs indiquant respectivement le début et la fin de la balise. La balise de fin de paragraphe sera `</para>`.

2.2. Différences entre SGML et XML

Comme évoqué précédemment, XML a éliminé une grande partie de la fonctionalité générale de SGML (comme décrite dans [38, 11, 12]). On notera, en particulier, les points suivants[1] :

- l'omission de balises est interdite, c'est-à-dire : chaque balise de début *doit* avoir une balise de fin correspondante ;
- pour les élément vides il y a une nouvelle notation `<a/>` (`<a></a>` est aussi valable) ;
- les commentaires ont nécessairement la forme `<!-- ... -->`, c'est-à-dire : il est impossible d'inclure des commentaires à l'intérieur des déclarations d'éléments ou d'attributs ;
- dans les modèles de contenu on ne peut utiliser des exclusions ou inclusions, ni l'opérateur *et* (`&`) ;
- toutes les valeurs des attributs sont spécifiées entre une paire de simples (`'`) ou doubles (`"`) *quotes* ;
- pour le nom des éléments et attributs, les majuscules et minuscules sont traitées comme des caractères distincts, donc `<Para>`, `<para>` et `<PARA>` sont des balises de trois éléments différents (en HTML classique la casse des caractères ne différencie pas les noms dans ce cas).

3. Anatomie d'une DTD pour un document XML

Comme nous l'avons indiqué ci-dessus une application XML peut traiter un document XML sans nécessairement avoir accès à la DTD décrivant la classe de documents pour le document en question. Néanmoins, il est primordial pour une bonne compréhension des possibilités du langage XML de rappeler la fonction et la structure de base d'une DTD. Il est à noter que la DTD peut être spécifiée en entier ou en partie à l'extérieur ou à l'intérieur d'un document source XML.

Une déclaration de type de document définit :

- le *nom* des types d'élément, leur *contenu*, y compris *combien de fois* et dans quel *ordre* d'autres éléments peuvent y apparaître (section 3.1) ;
- les *attributs* éventuels et leur valeurs par défaut (section 3.2) ;
- le nom des *entités* qui peuvent être utilisées (section 3.3).

Comme la précision et la robustesse de l'information balisée en XML est un atout non négligeable, il est important de pouvoir vérifier les documents XML par rapport à une

1. Une liste plus complète des différences entre SGML et XML se trouve à l'URL `http://www.w3.org/TR/NOTE-sgml-xml` et dans l'article de Sarra BEN LAGHA dans ce *Cahier*, page 127

DTD. Ainsi, nous serons assurés de leur validité et nous pourrons les distribuer sur le web sans poser de problèmes aux applications qui pour des raison d'efficacité les interpréteraient sans référence à la DTD.

Il est utile de répéter qu'XML différencie la casse des caractères (présence de minuscules et majuscules) dans les noms des éléments (balises), des attributs et des entités, qui doivent donc toujours être entrés textuellement.

3.1. Éléments

Chaque type d'élément de la structure logique d'un document doit être déclaré. Cette déclaration donne le nom du type de l'élément et indique, entre parenthèses, ce que peut, ou doit contenir l'élément en question (son *modèle de contenu*). Un exemple est le suivant :

```
<!ELEMENT mémo  (entête, contenu, ps?)>
```

Ce modèle indique que l'élément « `mémo` » doit contenir un élément `entête`, suivi d'un `contenu`, puis peut (sans obligation) être suivi d'un élément `ps`.

Le modèle de contenu peut faire appel à des opérateurs d'ordre et de choix, comme suit :

`a | b`	`a` *ou* `b` (dans n'importe quel ordre) ;
`a , b`	`a` *suivi* de `b` (dans l'ordre indiqué) ;
`a`	une fois l'élément `a` ;
`a?`	zéro ou une fois l'élément `a` ;
`a*`	zéro ou plusieurs fois l'élément `a` ;
`a+`	au moins une fois l'élément `a` ;
`(...)`	un *groupe* d'éléments.

Un élément peut être *vide* (mot clé `EMPTY`), il peut contenir d'autres éléments, des caractères[2], ou un mélange des deux : dans ce dernier cas on parle de *contenu mixte*. Pour un contenu mixte XML est très strict : ni le nombre ni l'ordre des éléments ne peuvent être spécifiés explicitement et le mot clé `#PCDATA` doit être le premier dans le modèle de contenu :

```
1 <!ELEMENT br EMPTY>
2 <!ELEMENT livre (chapitre+,annexe*,biblio?,index?>
```

2. Des données de type « caractère » sont celles déjà traitées par le programme d'analyse (en anglais *parser*) et qui ne contiennent plus d'appels d'entités ou de balises. Dans le jargon XML cela s'appelle *parsed character data* ou données textuelles analysées et se note `#PCDATA`.

```
3 <!ELEMENT e1 (#PCDATA)>
4 <!ELEMENT e2 (#PCDATA|a1|a2|a3)*>
5 <!ELEMENT e3 (#PCDATA | %ent1; | %ent2;)* >
```

La première ligne déclare l'élément `br` comme étant vide. La deuxième ligne déclare qu'un livre contient au moins un élément `chapitre`, puis aucun ou plusieurs éléments `annexe`, puis aucun ou un élément `biblio` suivi d'un éventuel élément `index`. La ligne 3 déclare que l'élément `e1` ne contient que des données textuelles analysées, alors que la ligne 4 déclare que l'élément `e2` peut contenir des données textuelles analysées, ainsi que les éléments `a1`, `a2` et `a3` (toutefois nous ne savons rien concernant le nombre d'occurrences, ni de leur ordre). La dernière ligne, qui fait référence à des entités paramètres, sera expliquée à la section 3.3.

Finalement, si un élément peut contenir des caractères ou tout autre élément défini dans la DTD nous utilisons le mot-clé `ANY`. Ce type de déclaration n'est pas très riche en information pour limiter la structure d'un document, mais a son utilité comme élément « conteneur » générique.

```
<!ELEMENT container ANY>
```

3.2. Attributs

Les attributs permettent d'associer des paires nom-valeur aux éléments définis dans la DTD. Des attributs peuvent seulement être spécifiés à l'intérieur d'une balise de début ou d'élément vide. Les définitions d'attribut spécifient :

- la liste exhaustive des noms d'attribut associés à un élément ;
- le type des attributs en question ;
- une valeur par défaut.

Une déclaration d'attribut comporte :

- le nom de l'élément ;
- le nom de l'attribut ;
- soit le *type de l'attribut*, indiqué par un mot-clé :

`CDATA`	données textuelles (caractères quelconques) ;
`ENTITY`	nom d'entité générale ;
`ENTITIES`	noms d'entités générales ;
`ID`	identificateur *unique* d'un élément ;
`IDREF`	valeur d'appel d'identificateur d'élément ;
`IDREFS`	valeurs d'appel d'identificateurs d'élément ;
`NMTOKEN`	unité lexicale nominale ;
`NMTOKENS`	unités lexicales nominales ;
`NOTATION`	nom de notation.

Alternativement on spécifie, entre parenthèses, la liste de toutes les valeurs possibles pour cet attribut ;
– une valeur par défaut, sous la forme d'une des valeurs autorisées, spécifiée entre guillemets, ou d'un mot-clé :

`#FIXED`	l'attribut a une valeur fixe, et ne peut prendre que cette valeur ;
`#REQUIRED`	une valeur doit obligatoirement être spécifiée par l'utilisateur ;
`#IMPLIED`	si une valeur n'est pas spécifiée, le système de traitement en définira une.

Voici un exemple, qui combine quelques-unes des possibilités évoquées. Dans le cas de l'attribut `type` la DTD doit contenir une définition de type `NOTATION` pour chacune des valeurs mentionnées, avec un renvoi vers une ressource système ou publique pour traiter les données associées à l'élément en question (p. ex. un programme de visualisation capable de décoder le format graphique choisi).

```
<!ATTLIST image source  %URL    #REQUIRED
                id      ID      #REQUIRED
                nom     CDATA   #IMPLIED
                échelle "1.0"   #FIXED
                fond    (rouge|bleu|jaune)  "bleu"
                type    NOTATION (eps|gif|bmp|tiff) "eps">
```

3.3. Entités

Des entités peuvent être utiles dans plusieurs circonstances :

– la définition de caractères difficiles ou impossibles à saisir avec un clavier standard (caractères nationaux, symboles graphiques et mathématiques). Avec un *appel de caractère* nous spécifions directement le code Unicode du caractère en question, par exemple l'esperluette (`&`) est le caractère 38 en Ascii :

```
<!ENTITY amp CDATA "&#38;">
```

Pour un document bien-formé, XML prédéfinit les entités `amp` (`&`), `lt` (`<`), `gt` (`>`), `apos` (`'`) et `quot` (`"`). Par contre, pour tout document valide ces entités doivent nécessairement être déclarées dans la DTD. Notons qu'avec XML on a accès par défaut à tous les caractères définis par Unicode[35, 2].
– La définition de notations raccourcies pour des suites de caractères saisies fréquemment (entités générales) p. ex.

```
<!ENTITY Seuil "<em>&Eacute;ditions du Seuil</em>">
```

– L'inclusion de fichiers externes, par exemples les différentes parties d'un long documents XML, ou des documents non-XML (entités externes) p. ex.

```
<!ENTITY poème3 SYSTEM "clair-de-lune.xml">
```

– La définition de variables dans une DTD pour en faciliter la structuration logique (entités paramètres) p. ex.

```
<!ENTITY % préambule  " titre | recueil | date | auteur ">
```

Les entités sont déclarées dans la DTD (dans le sous-ensemble interne ou externe). Comme nous l'avons montré dans les exemples ci-dessus, la définition d'une entité peut contenir des balises, des appels d'entités, etc., qui seront interprétés au moment où l'entité sera développée par le programme d'analyse XML.

Pour faire référence à une entité générale on utilise un *appel d'entité*, qui a la forme `&` suivi du nom de l'entité suivi de `;` . Par exemple pour utiliser l'entité `Seuil` définie ci-dessus nous saisissons dans le texte `&Seuil;` ce qui générera la suite de caractères « `<em>&Eacute;ditions du Seuil</em>` ».

Les entités paramètres, qui peuvent seulement être définies et utilisées à l'intérieur de la DTD sont un outil commode pour modulariser la structure d'une DTD et ainsi la rendre plus lisible et plus facile à tenir à jour ou à modifier. Tout définition d'une entité paramètre doit précéder son utilisation.

3.4. Exemple : un poème de Verlaine

Dans ce qui suit nous allons utiliser plusieurs types de documents, et chaque fois leur structure sera décrite à l'aide d'une ou de plusieurs DTD. Supposons que nous voulions créer une DTD pour saisir des poèmes. Comme exemple de texte nous utiliserons la célèbre « Chanson d'automne » de Paul Verlaine (du recueil *Poèmes saturniens* édité en 1866). Regardons le texte en question.

Les sanglots longs
Des violons
  De l'automne
Blessent mon coeur
D'une langueur
  Monotone.
Tout suffocant
Et blême, quand
  Sonne l'heure,

Je me souviens
Des jours anciens
  Et je pleure ;
Et je m'en vais
Au vent mauvais
  Qui m'emporte
Deçà, delà,
Pareil à la
  Feuille morte.

3.4.1. Première réalisation : les éléments triomphent

Nous voulons pouvoir spécifier le titre du poème, son auteur, peut-être le recueil dont il a été tiré et sa date d'édition. Tout mettrons tout cela un préambule. Pour le corps d'un poème nous préconisons de le diviser en stances qui, à leur tour, contiennent

des lignes, avec ou sans renfoncement. La DTD ci-dessous est une première réalisation du cahier de charges évoqué où nous codons toute l'information à l'aide d'éléments.

La seule partie dans cette DTD que nous n'avons pas encore rencontrée est la première ligne. Elle est optionnelle et correspond à la *déclaration* XML. Elle spécifie la version de XML utilisée (mot-clé `version`) qui peut être suivie du codage (mot-clé `encoding`) et une spécification si le document peut être traité seul, c'est-à-dire : sans devoir faire appel à des déclarations d'éléments dans une DTD externe, auquel cas on ajoute « `standalone="yes"` », ou sinon « `standalone="no"` ».

Voici le fichier source de la DTD, `peomfr.dtd` :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- poemfr.dtd : DTD pour poésie    M. Goossens  -->
<!-- poème contient un préambule suivi d'un corps -->
<!ELEMENT poème      (préambule, corps)>
<!-- préambule doit contenir un titre,
     peut-être suivi du nom du recueil,
     peut-être suivi de la date de composition,
     doit être suivi du nom de l'auteur -->
<!ELEMENT préambule  (titre, recueil?, date?, auteur)>
<!-- titre contient des caractères -->
<!ELEMENT titre      (#PCDATA)>
<!-- recueil contient des caractères -->
<!ELEMENT recueil    (#PCDATA)>
<!-- date contient des caractères -->
<!ELEMENT date       (#PCDATA)>
<!-- auteur contient des caractères -->
<!ELEMENT auteur     (#PCDATA)>
<!-- corps a des lignes, regroupées ou non en stances -->
<!ELEMENT corps      (stance|ligne)+>
<!-- stance composée de lignes -->
<!ELEMENT stance     (ligne)+>
<!-- ligne contient caractères et (peut-être) renfoncements -->
<!ELEMENT ligne      (#PCDATA|r)*>
<!-- renfoncement contient des caractères -->
<!ELEMENT r          EMPTY>
```

Maintenant nous pouvons baliser notre poème en utilisant cette DTD. Nous montrons le fichier source XML `verlainefr.xml` ci-dessous. La deuxième ligne y correspond à une *déclaration de type de document* (mot-clé `DOCTYPE`). Dans ce cas elle spécifie que le document a comme racine l'élément `poème` dont la structure est définie dans une DTD qui se trouve dans le fichier `poemfr.dtd`, qui est disponible sur le système (mot-clé `SYSTEM`) où le document est traité.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE poème SYSTEM "poemfr.dtd">
<poème>
<préambule>
<titre>Chanson d'automne</titre>
<recueil>Poèmes saturniens</recueil>
<date>1866</date>
<auteur>Paul Verlaine</auteur>
```

```
</préambule>
<corps>
<stance>
<ligne>Les sanglots longs</ligne>
<ligne>Des violons</ligne>
<ligne><r/>De l'automne</ligne>
<ligne>Blessent mon coeur</ligne>
<ligne>D'une langueur</ligne>
<ligne><r/>Monotone.</ligne>
</stance>
<stance>
<ligne>Tout suffocant</ligne>
<ligne>Et blême, quand</ligne>
<ligne><r/>Sonne l'heure,</ligne>
</stance>
<stance>
<ligne>Je me souviens</ligne>
<ligne>Des jours anciens</ligne>
<ligne><r/>Et je pleure ;</ligne>
</stance>
<stance>
<ligne>Et je m'en vais</ligne>
<ligne>Au vent mauvais</ligne>
<ligne><r/>Qui m'emporte</ligne>
<ligne>Deçà, delà,</ligne>
<ligne>Pareil à la</ligne>
<ligne><r/>Feuille morte.</ligne>
</stance>
</corps>
</poème>
```

À l'aide d'un programme d'analyse nous pouvons vérifier si le fichier XML est valide. Nous le faisons, par exemple, avec `xml4j` [21], un programme Java développé par IBM qui peut valider des documents XML.

```
> java XJParse -p com.ibm.xml.parsers.ValidatingSAXParser verlainefr.xml
verlainefr.xml: 460 ms (35 elems, 0 attrs, 49 spaces, 300 chars)
```

Le même programme offre la possibilité d'obtenir une vue en arbre du document.

```
> java ui.TreeViewer verlainefr.xml
verlainefr.xml:
START createUI:verlainefr.xml
START refreshUI:verlainefr.xml
START getRoot:verlainefr.xml
START readXMLFileverlainefr.xml
END readXMLFileverlainefr.xml
END refreshUI:verlainefr.xml
END createUI:verlainefr.xml
```

Le résultat est montré à la figure 1. Nous y voyons dans la fenêtre de gauche les différentes parties de l'arbre, que l'on peut traverser facilement en ouvrant ou fermant certains éléments (visibles comme dossier). À droite nous avons le fichier source.

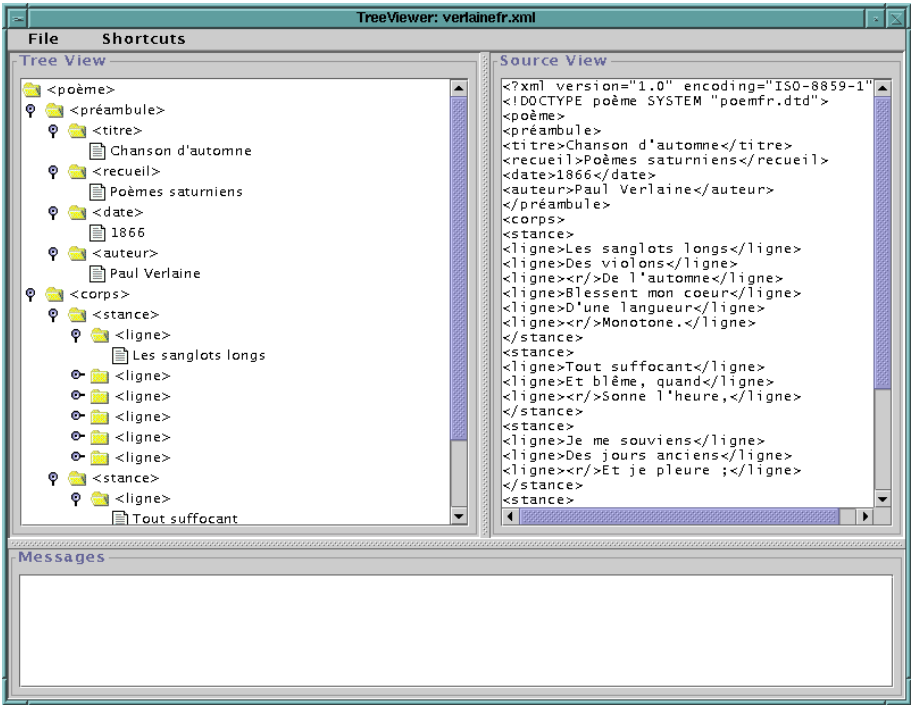


FIGURE 1 – *Représentation en arbre du document* `verlainefr.xml`

Nous pouvons aussi traiter ce fichier avec le programme d'analyse SGML `nsgmls`[6] de James CLARK et obtenir une sortie esis[7]. Nous devons toutefois indiquer qu'il s'agit d'un fichier XML, d'où la présence du fichier `xml.dcl` (la déclaration SGML adaptée pour XML) sur la ligne de commande ci-dessous.

```
> nsgmls xml.dcl verlainefr.xml
```

```
1  (poème
2  (préambule
3  (titre
4  -Chanson d'automne
5  )titre
6  (recueil
7  -Poèmes saturniens
8  )recueil
9  (date
10 -1866
11 )date
12 (auteur
13 -Paul Verlaine
14 )auteur
15 )préambule
16 (corps
17 (stance
18 (ligne
19 -Les sanglots longs
20 )ligne
21 (ligne
22 -Des violons
23 )ligne
24 (ligne
25 (r
26 )r
27 -De l'automne
28 )ligne
29 (ligne
30 -Blessent mon coeur
31 )ligne
32 (ligne
33 -D'une langueur
34 )ligne
35 (ligne
36 (r
37 )r
38 -Monotone.
39 )ligne
40 )stance
41 (stance
42 (ligne
43 -Tout suffocant
44 )ligne
45 (ligne
```

```
-Et blême, quand
)ligne
(ligne
(r
)r
-Sonne l'heure,
)ligne
)stance
(stance
(ligne
-Je me souviens
)ligne
(ligne
-Des jours anciens
)ligne
(ligne
(r
)r
-Et je pleure ;
)ligne
)stance
(stance
(ligne
-Et je m'en vais
)ligne
(ligne
-Au vent mauvais
)ligne
(ligne
(r
)r
-Qui m'emporte
)ligne
(ligne
-Deçà, delà,
)ligne
(ligne
-Pareil à la
)ligne
(ligne
(r
)r
-Feuille morte.
)ligne
)stance
)corps
)poème
C
```

La signification des différentes composantes de cette sortie esis est détaillée à l'URL `http://www.jclark.com/sp/sgmlsout.htm`. Pour nous, il suffit de savoir que les parenthèses représentent les éléments et qu'ils sont étiquetés du nom de l'élément en question. Le contenu de l'élément est contenu entre la parenthèse ouverte et la parenthèse fermée (par exemple le contenu de l'élément `préambule` commence à la ligne 2 et finit à la ligne 15). Les attributs commencent par un `A` majuscule (voir fichier esis à la page 15).

3.4.2. Deuxième variante : un savant mélange d'éléments et d'attributs

En général il y a plusieurs manières pour approcher la définition d'une DTD, qui décrit la façon dont nous voulons structurer les données dans notre document XML. Une approche assez traditionnelle balisant pratiquement toute l'information à l'aide d'éléments a été présentée. Nous pouvons tout aussi bien préférer de baliser uniquement le texte du poème à l'aide d'éléments et de spécifier tous les autres détails avec des attributs. C'est l'approche adoptée dans notre deuxième variante de DTD que nous avons baptisée `poemfr1.dtd` et qui a la forme suivante :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- poemfr1.dtd : DTD pour poésie    M. Goossens -->
<!-- poème contient un préambule suivi d'un corps -->
<!ELEMENT poème      (préambule, corps)>
<!-- préambule : élément vide                    -->
<!ELEMENT préambule  EMPTY>
<!-- attributs de type caractères pour titre, nom du recueil  -->
<!--                        date d'édition et nom de l'auteur -->
<!ATTLIST préambule titre   CDATA #IMPLIED
                    recueil CDATA #IMPLIED
                    date    CDATA #IMPLIED
                    auteur  CDATA #IMPLIED>
<!-- le corps contient des lignes -->
<!ELEMENT corps      (ligne)+>
<!-- ligne contient caractères -->
<!ELEMENT ligne      (#PCDATA)*>
<!-- ligne peut commencer nouvelle stance ou contenir renfoncement -->
<!ATTLIST ligne stance (oui|non)  "non"
                r      CDATA      "0em" >
```

Voici le fichier `verlainefr1.xml` qui contient du poème balisé d'après cette DTD.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE poème SYSTEM "poemfr1.dtd" [
3  <!ENTITY r "6pt">
4  ]>
5
6  <poème>
7  <préambule titre="Chanson d'automne"
8             recueil="Poèmes saturniens"
9             date="1866"
10            auteur="Paul Verlaine"/>
11 <corps>
12 <ligne stance="oui">Les sanglots longs</ligne>
13 <ligne>Des violons</ligne>
14 <ligne r="1em">De l'automne</ligne>
15 <ligne>Blessent mon coeur</ligne>
16 <ligne>D'une langueur</ligne>
17 <ligne r=".5em">Monotone.</ligne>
18 <ligne stance="oui">Tout suffocant</ligne>
19 <ligne>Et blême, quand</ligne>
20 <ligne r="&r;">Sonne l'heure,</ligne>
21 <ligne stance="oui">Je me souviens</ligne>
22 <ligne>Des jours anciens</ligne>
23 <ligne r="6pt">Et je pleure ;</ligne>
24 <ligne stance="oui">Et je m'en vais</ligne>
25 <ligne>Au vent mauvais</ligne>
26 <ligne r="&r;">Qui m'emporte</ligne>
27 <ligne>Deçà, delà,</ligne>
28 <ligne>Pareil à la</ligne>
29 <ligne r="1em">Feuille morte.</ligne>
30 </corps>
31 </poème>
```

Les informations concernant auteur, titre, etc., (lignes 7–10) ainsi que les débuts des stances (lignes 12, 18, 21, 24) et les renfoncements éventuels (lignes 14, 17, etc.) sont ici codés à l'aide d'attributs. Nous vérifions la validité du fichier avec `xml4j`.

```
> java XJParse -p com.ibm.xml.parsers.ValidatingSAXParser verlainefr1.xml
verlainefr1.xml: 499 ms (21 elems, 40 attrs, 36 spaces, 249 chars)
```

La figure 2 est une représentation par l'application `xmlviewer`[20] du document `verlainefr1.xml`. Une vue en arbre est dans la fenêtre en haut à gauche, les attributs de l'élément mis en évidence à droite, puis on trouve le document source en bas à gauche et finalement la DTD à droite. Cette application est très intéressante pour naviguer à travers un document XML.

```
java  com.ibm.xml.xmlviewer.XmlViewer verlainefr1.xml
```

Nous montrons également une partie de la sortie esis obtenue avec le programme `nsgmls`.

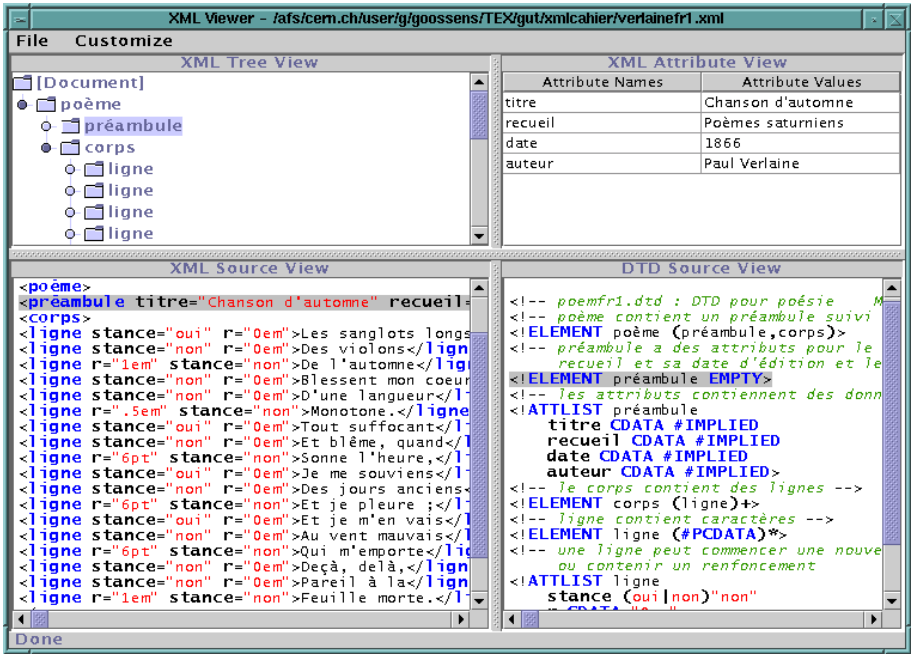


FIGURE 2 – *Le document* `verlainefr1.xml` *vu par l'application* `xmlviewer`

```
(poème
Atitre CDATA Chanson d'automne
Arecueil CDATA Poèmes saturniens
Adate CDATA 1866
Aauteur CDATA Paul Verlaine
(préambule
)préambule
(corps
Astance TOKEN oui
Ar CDATA 0em
(ligne
-Les sanglots longs
)ligne
Astance TOKEN non
Ar CDATA 0em
(ligne
-Des violons
)ligne
   ...
Astance TOKEN non
Ar CDATA 6pt
(ligne
-Qui m'emporte
)ligne
Astance TOKEN non
Ar CDATA 0em
(ligne
-Deçà, delà,
)ligne
Astance TOKEN non
Ar CDATA 0em
(ligne
-Pareil à la
)ligne
Astance TOKEN non
Ar CDATA 1em
(ligne
-Feuille morte.
)ligne
)corps
)poème
C
```

Nous y observons clairement la présence des attributs et de leur valeur (par exemple la ligne 2 indique que `titre` est de type `CDAT` et a la valeur « Chanson d'automne »). Notons également que les valeurs par défaut sont utilisées pour les attributs non spé-

cifiés explicitement (ainsi en ligne 14 `stance` a la valeur `non` et en ligne 15 `r` une valeur égale a 0em).

3.4.3. Variante ultime : rien que des attributs

Nous pouvons même aller à l'extrême et spécifier tout le poème comme valeurs d'attributs. Cette ultime approche montre également comment étendre une DTD en spécifiant des attributs supplémentaires à ceux déjà définis dans le sous-ensemble *externe* de la DTD. Ceci se fait dans le sous-ensemble *interne* de la DTD, c'est-à-dire : dans la partie *déclaration de balisage* à l'intérieur de la *déclaration de type de document*. C'est ce qui est montré ci-dessous en lignes 3–8. XML permet plusieurs instructions pour les définitions d'attributs `<!ATTLIST >` pour un élément donné : toutes les définitions sont combinées. On peut même donner plusieurs définitions pour le *même* attribut pour un élément donné. Toutefois dans ce cas, seule la *première* définition est prise en compte et les définitions ultérieures sont ignorées. Comme le sous-ensemble interne de la DTD est lu *avant* le sous-ensemble externe, dans notre cas les définitions en lignes 3–7 dans le fichier `verlainefr2.xml` montré ci-dessous sont lues avant celles dans le fichier DTD `poemfr1.dtd` référencé en ligne 2.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE poème SYSTEM "poemfr1.dtd" [
3  <!ENTITY r "1em">
4  <!-- le numéro de la ligne (défaut : pas de numéro) -->
5  <!ATTLIST ligne nb     CDATA " "       >
6  <!-- le texte de la ligne  (défaut : pas de texte) -->
7  <!ATTLIST ligne texte  CDATA " "       >
8  ]>
9  <poème>
10 <préambule titre="Chanson d'automne"
11               recueil="Poèmes saturniens"
12               date="1866"
13               auteur="Paul Verlaine"/>
14 <corps>
15 <ligne stance="oui" nb="1"  texte="Les sanglots longs"/>
16 <ligne              nb="2"  texte="Des violons"/>
17 <ligne r="&r;"      nb="3"  texte="De l'automne"/>
18 <ligne              nb="4"  texte="Blessent mon coeur"/>
19 <ligne              nb="5"  texte="D'une langueur"/>
20 <ligne r="&r;"      nb="6"  texte="Monotone."/>
21 <ligne stance="oui" nb="7"  texte="Tout suffocant"/>
22 <ligne              nb="8"  texte="Et blême, quand"/>
23 <ligne r="&r;"      nb="9"  texte="Sonne l'heure,"/>
24 <ligne stance="oui" nb="10" texte="Je me souviens"/>
25 <ligne              nb="11" texte="Des jours anciens"/>
26 <ligne r="&r;"      nb="12" texte="Et je pleure ;"/>
27 <ligne stance="oui" nb="13" texte="Et je m'en vais"/>
28 <ligne              nb="14" texte="Au vent mauvais"/>
29 <ligne r="&r;"      nb="15" texte="Qui m'emporte"/>
30 <ligne              nb="16" texte="Deçà, delà,"/>
31 <ligne              nb="17" texte="Pareil à la"/>
32 <ligne r="&r;"      nb="18" texte="Feuille morte."/>
33 </corps>
34 </poème>
```

Nous avons balisé l'élément `<ligne>` comme s'il était élément vide, même si, dans la DTD, il n'a pas été défini comme tel. Ce balisage explicite et complet est très commode pour gérer des textes dans les bases de données, comme il est simple d'en extraire n'importe quel type d'information. Évidemment pour chaque DTD une nouvelle déclaration XSL devra être développée pour visualiser le document (voir la section 5).

4. Domaines nominaux

Les domaines nominaux (*namespaces*[41]) permettent de garantir l'unicité des noms d'éléments et d'attributs même lorsque plusieurs DTD (modules ou logiciels) sont combinées. En effet il n'est pas rare que des noms identiques soient utilisés dans ces modules à des fins différentes. La spécification définissant les domaines nominaux introduit une méthode pour délimiter la partie du document où un nom est visible en créant un « domaine de validité » comme suit :

- un domaine nominal est un ensemble de noms caractérisé par un URI [22] ;
- cet URI identifie d'une façon unique les noms de types d'élément et d'attribut appartenant à ce domaine nominal ;
- cet URI ne doit pas correspondre à une ressource réelle ;
- cet URI servira comme préfixe pour rendre un nom non-ambigu.

Un domaine nominal est défini à l'aide d'une série d'attributs réservés. Un tel attribut sera la chaîne de caractères `xmlns` ou aura la chaîne `xmlns:` comme préfixe.

Le double point est utilisé pour séparer le préfixe identifiant le domaine nominal de la partie locale, les deux unités ensemble formant un « nom qualifié » (voir les règles de production à la section B pour plus de détails).

```
1 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
2                 xmlns:fo="http://www.w3.org/1999/XSL/Format">
3 <xsl:template match="emph">
4   <fo:sequence font-style="italic">
5     <xsl:apply-templates/>
6   </fo:sequence>
7 </xsl:template>
```

Dans l'exemple précédent nous voyons comment le préfixe `xmlns:` permet de définir l'abréviation qui sera utilisée pour identifier un domaine nominal. Ainsi « `xsl:` » en ligne 1 est déclaré correspondre à la chaîne `http://www.w3.org/1999/XSL/Transform`, alors que la ligne 2 déclare `fo:` comme représentant la chaîne `http://www.w3.org/XSL/Format/1.0`.

La partie locale du nom qualifié — `template` en ligne 3 ou `apply-templates` en ligne 5 et `sequence` en ligne 4 — doit avoir un sens dans le domaine nominal en question — `xsl` et `fo` dans notre cas.

Un domaine nominal *par défaut* peut être indiqué comme à la ligne 3 ci-dessous, qui spécifie que les noms non préfixés (comme le <p> en ligne 5) appartiennent au domaine nominal XHTML `http://www.w3.org/TR/xhtml1`.

```
1 <xsl:stylesheet
2  xmlns:xsl="http://www.w3.org/XSL/Transform/"
3  xmlns="http://www.w3.org/TR/xhtml1"
4 <xsl:template match="par">
5 <p><xsl:apply-templates/></p>
6 </xsl:template>
```

4.1. Déclarer un domaine nominal pour un élément

Une déclaration de domaine nominal est valable pour l'élément où elle est spécifiée et pour tous les éléments contenus dans cet élément, à moins qu'une autre déclaration de domaine nominal utilisant la même partie abréviation ne soit rencontrée. Ainsi, les domaines nominaux peuvent se chevaucher et être redéfinis localement comme le montre l'exemple suivant :

```
1  <?xml version='1.0'?>
2   ...
3   <particules>
4     <!-- Domaine nominal devient XHTML -->
5     <table xmlns='http://www.w3.org/TR/xhtml1'>
6      <tr><td>Particule</td><td>Masse</td><td>Détails</td></tr>
7      <tr>
8        <td><em>neutron</em></td>
9        <td>939.56 MeV</td>
10       <td>
11         <pdg xmlns="autreDomNom">
12          <!-- Ici le domaine nominal change localement  -->
13          <quarks>udd</quarks><duréeDeVie>886.7 s</duréeDeVie>
14          <désintégration>p,e,anue</désintégration>
15         </pdg>
16       </td><!-- À partir d'ici nous retrouvons XHTML -->
17     </tr>
18    </table>
19   <!-- Retour au domaine original -->
20   </particules>
21   ...
```

De la ligne 5 à la ligne 18, le domaine nominal par défaut est celui de XHTML. Par contre, en ligne 11 nous redéfinissons le domaine nominal par défaut qui devient `autreDomNom` pour l'élément `pdg` et ses descendants. Il reste effectif jusqu'en ligne 15 où nous quittons le cadre de l'élément `pdg`.

Si nous voulons éliminer toute référence à un domaine nominal nous pouvons spécifier un URI vide (`xmlns=""`). Dans ce cas les éléments sans préfixe nominal dans le domaine de validité de la déclaration n' appartiennent à aucun domaine nominal. Notons toutefois (voir exemple ci-dessous) que le domaine nominal par défaut ne s'applique par directement aux attributs.

4.2. Déclarer un domaine nominal pour un attribut

Les domaines nominaux s'appliquent aussi aux attributs. Notons qu'une balise XML ne peut contenir deux attributs avec un nom identique (en accord avec la section 3.1 de la spécification XML), ni avec des noms qualifiés ayant une même partie locale et des préfixes liés à des domaines nominaux identiques.

```
1 <ex xmlns:d1="http://www.gutenberg.eu.org"
2     xmlns:d2="http://www.gutenberg.eu.org" >
3     xmlns="http://www.gutenberg.eu.org" >
4   <faux a="1"     a="2" />
5   <faux d1:a="1"  d2:a="2" />
6   <bien a="1"     b="2" />
7   <bien a="1"     d1:a="2" />
8 </ex>
```

Dans cet exemple, les lignes 1–3 définissent trois domaines nominaux (`d1`, `d2` et le défaut) qui correspondent tous à l'URI `http://www.gutenberg.eu.org`. La ligne 4 est illégale (les deux attributs ont le même nom), tout comme la ligne 5 (les deux attributs ont une même partie locale et des domaines nominaux liés à un URL identique). Par contre la ligne 6 dont les deux attributs ont un nom différent est légale. La ligne 7 est également correcte car le domaine nominal par défaut ne s'applique pas aux attributs.

5. Le langage XSL

La spécification du langage XSL (*Extensible Stylesheet Language* ou langage extensible pour feuilles de style) est divisée en trois parties :

1. XPath[48] : un langage pour adresser une partie d'un document XML. Il est utilisé avec les recommandations XSLT et XPointer.
2. XSLT [53] : un langage pour transformer les documents XML en d'autres documents XML.
3. XSLF [52] : un vocabulaire pour spécifier le formatage d'un document XML. XSLF utilise une feuille de style XSLT pour transformer le document source XML en un autre document XML qui utilise les éléments du vocabulaire de formatage.

Actuellement la plupart des applications de XSL n'utilisent que les parties XPath et XSLT. Nous les traiterons en priorité et ne mentionnerons qu'en passant quelques caractéristiques de XSLF, dont la spécification n'est pas encore finalisée. En complément à l'information sur XSL dans ses pages nous suggérons des tutoriels, listes de discussions et foire aux questions dans [27, 26, 8, 25, 17].

5.1. Le langage XPath

Le langage XPath propose une syntaxe et sémantique communes pour les fonctions partagées entre XSLT et XPointer[49]. Le but principal de XPath est d'adresser les différentes parties d'un document XML. XPath propose également quelques fonctions pour manipuler les suites de caractères, les nombres et les booléens. La syntaxe du langage XPath adopte une syntaxe compacte non-XML, qui permet son utilisation dans les URIs et à l'intérieur des attributs XML.

XPath modélise un document XML comme un arbre de nœuds (de type élément, attribut, texte, etc..). XPath permet de calculer une valeur de type caractère pour chaque type de nœud. La construction syntaxique principale de XPath est l'*expression* dont l'évaluation peut donner quatre types d'objets : un ensemble (une collection non-ordonnée) de nœuds, un booléen (« vrai » ou « faux ») , un nombre (virgule flottante) ou une chaîne de caractères (une séquence de caractères Unicode).

5.1.1. Chemins de localisation

Un *chemin de localisation* (*location path*) est le type d'expression le plus commun de XPath. Il utilise treize *axes* pour se déplacer dans le document.

`child` contient les descendants immédiats (les « enfants ») du nœud contexte, en d'autres mots le nœud par rapport auquel on spécifie le chemin de localisation. Les nœuds de type attribut ou domaine nominal ne sont pas considérés comme des descendants du l'élément contenant.
`child::section` choisit les nœuds de type `section` qui sont des descendants immédiats du nœud contexte,
`child::*` choisit *tous* les descendants directs du nœud contexte.

`descendant` contient tous les descendants (enfants, petits-enfants, ...) du nœud contexte. Comme pour l'axe `child` cet axe ne contient jamais de nœuds de type attribut ou domaine nominal.
`descendant::soussection` choisit les nœuds de type `soussection` qui sont des descendants du nœud contexte.

`parent` contient l'ascendant immédiat (parent au sens « père » ou « mère ») du nœud contexte, s'il existe. Pour un nœud de type attribut et domaine nominal il s'agit de l'élément auquel le nœud en question est attaché (l'élément « contenant »).

`parent::article` choisit l'ascendant immédiat de type `article` du nœud contexte.

`ancestor` contient les ascendants (parent, grand-parent, etc.) du nœud contexte. Le parent sera le premier sur l'axe, le parent du parent le deuxième, etc.
`ancestor::article` choisit l'ascendant de type `article` du nœud contexte.

`following-sibling` contient les nœuds qui suivent le nœud contexte et qui ont le même parent que ce dernier (c'est-à-dire : les « frères » et « sœurs »). Pour un nœud de type attribut ou domaine nominal l'axe `following-sibling` est vide.

`preceding-sibling` contient les nœuds qui précèdent le nœud contexte et qui ont le même parent que ce dernier. Pour un nœud de type attribut ou domaine nominal l'axe `following-sibling` est vide.

`following` contient tous les nœuds dans le même document que le nœud contexte qui le suivent d'après l'ordre dans le document, en excluant les descendants et les nœuds de type attribut ou domaine nominal.

`preceding` contient tous les nœuds dans le même document que le nœud contexte qui le précèdent d'après l'ordre dans le document, en excluant les ascendants et les nœuds de type attribut ou domaine nominal.

`self` ne contient que le nœud contexte.

`descendant-or-self` contient le nœud contexte et ses descendants ; le nœud contexte est le premier nœud sur l'axe, son premier enfant le deuxième, et ainsi de suite (voir l'axe `descendant`).

`ancestor-or-self` contient le nœud contexte et ses ascendants ; le nœud contexte est le premier nœud sur l'axe, son parent le deuxième, et ainsi de suite (voir l'axe `parent`).

`attribute` contient les attributs du nœud contexte. Si le nœud contexte n'est pas de type élément l'axe sera vide.

`namespace` contient les nœuds de type domaine nominal du nœud contexte. L'axe est vide si le nœud contexte n'est pas de type élément.

Pour certains cas il existe une notation courte d'après le schéma suivant (la forme courte est à gauche, son équivalent long à droite) :

nom	`child::`*nom*
`@`*nom*	`attribute::`*nom*
`.`	`self::node()`
`..`	`parent::node()`
`//`	`descendant-or-self::node()`

Voici quelques exemples de l'utilisation de ces axes (avec la forme courte équivalente si elle existe) :

– `"child::section"` choisit les enfants de type `section` du nœud contexte. `"section"` en est la forme courte.

- `"child::text()"` choisit les nœuds enfants de type texte du nœud contexte. `"texte()"` en est la forme courte.
- `"child::node()"` choisit tous les nœuds enfants du nœud contexte indépendamment du type du nœud. `"node()"` en est la forme courte.
- `"attribute::couleur"` choisit l'attribut ayant le nom *couleur* du nœud contexte. `"@couleur"` en est la forme courte.
- `"attribute::*"` choisit tous les attributs du nœud contexte. `"@*"` en est la forme courte.
- `"descendant::soussection"` choisit les descendants de type `soussection` du nœud contexte.
- `"ancestor::soussection"` choisit les ascendants de type `soussection` du nœud contexte.
- `"ancestor-or-self::soussection"` choisit les ascendants de type `soussection` du nœud contexte ainsi que le nœud contexte lui-même s'il correspond au type d'élément `soussection`.
- `"descendant-or-self::soussection"` choisit les descendants de type `soussection` du nœud contexte ainsi que le nœud contexte lui-même s'il correspond au type d'élément `soussection`. `"//soussection"` en est la forme courte.
- `"self::soussection"` choisit le nœud contexte s'il correspond au type d'élément `soussection`, autrement ne choisit rien.
- `"child::section/descendant::soussection"` choisit les descendants de type `soussection` des enfants de type `section` du nœud contexte. `"section//soussection"` en est la forme courte.
- `"child::*/child::para"` choisit tous les petits-enfants de type `para` du nœud contexte.
- `"/"` sélectionne la racine du document (qui est toujours le parent de l'élément `document` (voir la production `[1]` de la grammaire de XML).
- `"/descendant::liste/child::item"` choisit tous les éléments de type `item` dans le même document que le nœud contexte qui ont un parent de type `liste` (le `/` initial indique que nous commençons à la racine du document).
- `"child::liste[position()=2]"` choisit le deuxième enfant de type `liste` du nœud contexte. `"liste[2]"` en est la forme courte.
- `"child::liste[position()=last()]"` choisit le dernier enfant de type `liste` du nœud contexte. `"liste[last()]"` en est la forme courte.
- `"child::liste[position()=last()-1]"` choisit l'avant-dernier enfant de type `liste` du nœud contexte. `"liste[last()-1]"` en est la forme courte.
- `"child::liste[position()=>3]"` choisit tous les enfants de type `liste` du nœud contexte sauf les deux premiers. `"liste[position()=>3]"` en est la forme courte.
- `"following-sibling::section[position()=1]"` choisit le prochain élément de type `section` qui a le même parent que le nœud contexte.

- `"preceding-sibling::section[position()=1]"` choisit le précédent élément de type `section` qui a le même parent que le nœud contexte.
- `"/descendant::figure[position()=17]"` choisit le dix-septième élément de type `figure` dans le document.
  `"//figure[17]"` en est la forme courte.
- `"/child::article/child::section[position()=3]/child::soussection[position()=5]"` choisit la cinquième `soussection` de la troisième `section` de l'élément document `article`.
  `"article/section[3]/soussection[5]"` en est la forme courte.
- `"child::note[attribute::type='attention']"` choisit les enfants du nœud de contexte de type `note` qui ont un attribut `type` qui a la valeur `attention`. `"note[@type='attention']"` en est la forme courte.
- `"child::note[attribute::type='attention'][position()=3]"` choisit le troisième enfant du nœud de contexte qui est de type `note` et qui a un attribut `type` qui a la valeur `attention`.
  `"note[@type='attention'][3]"` en est la forme courte.
- `"child::note[position()=3][attribute::type='attention']"` choisit le troisième enfant du nœud de contexte s'il est de type `note` et a un attribut `type` qui a la valeur `attention`.
  `"note[3][@type='attention']"` en est la forme courte.
- `"child::section[child::titre]"` choisit les enfants du nœud de contexte qui sont de type `section` et qui ont un ou plusieurs enfants `titre`.
  `"section[titre]"` en est la forme courte.
- `"child::section[child::titre='Introduction']"` choisit les enfants du nœud de contexte qui sont de type `section` et qui ont un ou plusieurs enfants `titre` avec une valeur « `Introduction` ».
  `"section[titre='Introduction']"` en est la forme courte.
- `"child::*[self::section or self::annexe]"` choisit les enfants de type `section` et `annexe` du nœud de contexte.
- `"child::*[self::section or self::annexe][position()=last()]"` choisit le dernier enfant de type `section` ou `annexe` du nœud de contexte.
- `"child::ligne[attribute::couleur and attribute::largeur]"` choisit les enfants de type `ligne` du nœud de contexte qui ont les deux attributs `couleur` et `largeur`.
  `"ligne[@couleur and @largeur]"` en est la forme courte.
- `"/descendant::para[1]"` choisit le premier descendant de type `para`.
- `"/descendant-or-self::node()/child::para[position()=1]"` choisit tous les éléments de type `para` qui sont les premiers enfants `para` de leur parents. `"//para[1]"` en est la forme courte.
- `"self::node()/descendant-or-self::node()/child::para"` choisit tous les éléments descendants de type `para` du nœud de contexte.
  `".//para"` en est la forme courte.

– `"parent::node()/child::titre"` choisit tous les enfants de type `titre` de l'élément parent du nœud de contexte.
`"../titre"` en est la forme courte.

Chaque axe possède un type de nœud principal :

– pour l'axe `attribute` le type de nœud principal est `attribut` ;
– pour l'axe `namespace` le type de nœud principal est `namespace` ;
– pour les autres types d'axe le type de nœud principal est « élément ».

Dans les exemples précédents nous avons implicitement introduit deux autres notions de base de XPath, à savoir la façon de spécifier les tests pour sélectionner un ensemble de nœuds et l'utilisation de prédicats pour qualifier davantage cette sélection.

La sélection est opérée en spécifiant une expression joker (*wildcard*) en combinaison le cas échéant avec une spécification d'un type de nœud, dont il existe quatre types : `comment()` (commentaire), `text()` (texte), `processing-instruction(Litéral)` (instruction de traitement), `node()` (type générique).

En particulier, le test nodal `child::section` choisit les éléments de type `section` qui sont des enfants du nœud contexte. En l'absence d'un tel élément, l'ensemble vide est sélectionné. D'une façon analogue le test nodal `attribute::couleur` choisit l'attribut `couleur` du nœud contexte. Si le nœud contexte ne possède pas un tel attribut l'ensemble vide est sélectionné.

Les expressions prédicats sont spécifiées entre crochets `[ ... ]`, ce qui donne par exemple : `[titre='Introduction']`.

5.1.2. Expressions

Une expression peut être une référence à une variable, une sous-expression, une valeur littérale, un nombre ou un appel de fonction. Des parenthèses peuvent être utilisées pour améliorer la compréhension ou pour guider l'analyse de l'expression.

Pour les appels de fonctions on évalue d'abord leurs arguments en les convertissant au type requis par la fonction, après quoi la fonction concernée est appelée dans la librairie des fonctions qui fait partie du contexte de l'évaluation de l'expression en lui passant les arguments convertis. Le résultat retourné par la fonction devient la valeur de la sous-expression.

Un chemin de localisation peut également figurer dans une expression. Dans ce cas l'expression retourne l'ensemble des nœuds sélectionnés par le chemin. L'opérateur d'union `|` combine plusieurs ensembles de nœuds.

La définition formelle de XPath se trouve à l'annexe C. Une liste des fonctions XPath et XSLT est à l'annexe E.

5.2. Introduction au langage XSLT

Une transformation XSLT — appelée une *feuille de style* — est une série de règles pour transformer un arbre source (*source tree*) XML en un arbre résultant (*result tree*). La transformation se fait en associant des motifs (*patterns*) à des modèles (*templates*) et en les appliquant aux éléments de l'arbre source. Chaque élément dans le fichier source ne peut correspondre qu'à un seul motif dans la feuille de style. Le modèle en question est alors initialisé pour transformer l'élément source en une partie de l'arbre résultant.

L'arbre source et l'arbre résultant sont disjoints et peuvent avoir une structure totalement différente. En effet des éléments de l'arbre source peuvent passer par un filtre, être réordonnés ou une structure arbitraire peut être insérée dans l'arbre résultant.

Comme indiqué ci-dessus une feuille de style contient une série de règles. Chaque règle a deux parties : un motif de correspondance (*match pattern*) pour sélectionner des éléments dans l'arbre source et un modèle qui sera initialisé pour construire une partie de l'arbre résultant. Cette structure générale permet d'appliquer une même feuille de style à une large classe de documents caractérisés par une arborescence source similaire. Exemple simple :

```
1 <xsl:template match="emph">         <!-- motif -->
2   <fo:sequence font-style="italic"> <!-- ++++++++++ -->
3     <xsl:apply-templates/>          <!-- + modèle + -->
4   </fo:sequence>                    <!-- ++++++++++ -->
5 </xsl:template>
```

La ligne un est le motif, qui spécifie que le modèle de traitement des lignes 2–4 est à appliquer aux éléments du document source qui ont un type `emph` (notons toutefois que certains éléments de type `emph` peuvent être traités différemment s'ils sont sélectionnés par un motif ayant une *priorité* plus élevée, voir la section 5.5 de la spécification XSLT[53]).

5.2.1. Structure d'une feuille de style

Ci-dessous suit une liste de tous les éléments qui peuvent être utilisés au niveau supérieur d'une feuille de style XSLT. Les endroits où le contenu ou un attribut doivent être spécifiés sont indiqués par `...`. Il est à souligner qu'une feuille de style peut contenir zéro ou plusieurs éléments des différents types indiqués. À part l'élément `<xsl:import ...>` (ligne 3), qui doit, s'il est présent, précéder tous les autres, l'ordre de spécification des éléments dans une feuille de style n'est pas significative, sauf

pour le traitement d'éventuelles conditions d'erreur. Notons que le domaine nominal `xsl:` (ligne 2) a été choisi pour préfixer tous les éléments XSLT.

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3    <xsl:import href="..."/>
4    <xsl:include href="..."/>
5    <xsl:strip-space elements="..."/>
6    <xsl:preserve-space elements="..."/>
7    <xsl:key name="..." match="..." use="..."/>
8    <xsl:functions ns="..."> ... </xsl:functions>
9    <xsl:attribute-set name="..."> ... </xsl:attribute-set>
10   <xsl:variable name="...">...</xsl:variable>
11   <xsl:param-variable name="...">...</xsl:param-variable>
12   <xsl:template match="..."> ... </xsl:template>
13   <xsl:template name="..."> ... </xsl:template>
14 </xsl:stylesheet>
```

5.2.2. Modèle de traitement

XSLT interprète un document XML comme un arbre, c'est-à-dire : deux feuilles de style ou documents source qui sont décrits par un même arbre seront traités par XSLT de façon identique.

XSLT connaît sept types de nœuds dans un arbre :

- le nœud *racine*, qui a comme unique descendant le nœud `document` ;
- les nœuds *élément*, qui correspondent aux éléments dans le source XML. Ils sont dans le même ordre que les balises de début dans le document source. Pour chaque attribut spécifié ou qui a une valeur par défaut un nœud attribut est créé. Un élément vide sans attributs n'a pas de nœuds attachés ;
- les nœuds *texte*, qui contiennent toutes les chaînes de caractères textuels (ne faisant pas partie du balisage). Toutes les données de type caractère qui se suivent sans balisage intermédiaire sont déposées dans un même contenant texte ;
- les nœuds *attribut*, qui contiennent des attributs associés à un élément et sont ainsi toujours attachés à un nœud élément ;
- les nœuds *domaine nominal*, qui sont construits à partir du document source et de la feuille de style ;
- les nœuds *instructions de traitement* (IT) qui sont créés pour chaque instruction de traitement avec un nom égal à la cible de l'IT. Un tel nœud a comme valeur la chaîne de caractères qui définit l'IT ;
- les nœuds *commentaire* qui sont générés pour chaque commentaire dans le fichier source.

À chaque type de nœud est associée une *valeur* (comme chaîne de caractères). Pour certains types cette valeur fait partie du nœud lui-même, alors que pour d'autres elle est calculée à partir de la valeur des nœuds des sous-éléments.

6. Exemples de feuilles de style XSL

Cette section contient des exemples de plus en plus complexe montrant l'utilisation des feuilles de style XSL pour transformer les documents XSL et obtenir différentes représentations. Pour bien comprendre les détails il faut se référer aux annexes C (définition du langage XPath), D (définition du langage XSLT) et E (liste des fonctions utilisés avec XPath et XSLT), ainsi qu'aux spécifications XPath[49] et XSLT [53].

6.1. Une invitation sous plusieurs formes

Notre premier exemple est emprunté au *LaTeX Web Companion*[15]. Nous avons décidé d'envoyer une invitation à nos collègues pour leur proposer de se voir quelque part pour faire la fête. Voici la DTD `invitation.dtd` que nous avons choisie pour modéliser l'information :

```
<!-- invitation DTD   -->
<!ELEMENT invitation (front, body, back) >
<!ELEMENT front      (to, date, where, why?) >
<!ELEMENT date       (#PCDATA) >
<!ELEMENT to         (#PCDATA) >
<!ELEMENT where      (#PCDATA) >
<!ELEMENT why        (#PCDATA) >
<!ELEMENT body       (par+) >
<!ELEMENT par        (#PCDATA|emph)* >
<!ELEMENT emph       (#PCDATA) >
<!ELEMENT back       (signature) >
<!ELEMENT signature  (#PCDATA) >
```

Avec ce modèle, qui n'utilise que des éléments pour indiquer le contenu sémantique du document XML, nous balisons le texte (anglais) `invitation.xml` suivant :

```
<?xml version="1.0"?>
<!DOCTYPE invitation SYSTEM "invitation.dtd">
<invitation>
<!-- ++++ The header part of the document ++++ -->
<front>
<to>Anna, Bernard, Didier, Johanna</to>
<date>Next Friday Evening at 8 pm</date>
<where>The Web Cafe</where>
<why>My first XML baby</why>
</front>
<!-- +++++ The main part of the document +++++ -->
<body>
<par>I would like to invite you all to celebrate
```

```
the birth of <emph>Invitation</emph>, my
first XML document child.</par>
<par>Please do your best to come and join me next Friday
evening. And, do not forget to bring your friends.</par>
<par>I <emph>really</emph> look forward to see you soon!</par>
</body>
<!-- +++ The closing part of the document ++++ -->
<back>
<signature>Michel</signature>
</back>
</invitation>
```

À l'aide de ce document nous allons montrer comment cette information peut être transformée en plusieurs formats (HTML, LaTeX, objets formatés).

6.1.1. Sortie LaTeX

Commençons par transformer le document, `invitation.xml` en LaTeX. Pour cette tâche nous nous basons sur la feuille de style XSL `invlat1.xsl` suivante :

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="text" indent="no" encoding="ISO-8859-1"/>

<xsl:strip-space elements="*"/>

<xsl:template match="invitation">
<xsl:text>\documentclass[12pt]{article}
\usepackage{invitation}
\begin{document}
</xsl:text>
<xsl:apply-templates/>
<xsl:text>\end{document}
</xsl:text>
</xsl:template>

<xsl:template match="front">
<xsl:text>\begin{Front}
\To{</xsl:text>
<xsl:value-of select="to"/>
<xsl:text>}
\Date{</xsl:text>
<xsl:value-of select="date"/>
<xsl:text>}
\Where{</xsl:text>
<xsl:value-of select="where"/>
<xsl:text>}
\Why{</xsl:text>
<xsl:value-of select="why"/>
<xsl:text>}
\end{Front}
</xsl:text>
</xsl:template>

<xsl:template match="body">
<xsl:text>\begin{Body}
</xsl:text>
```

```
39    <xsl:apply-templates/>
40  <xsl:text>
41  \end{Body}
42  </xsl:text>
43  </xsl:template>
44
45  <xsl:template match="par">
46  <xsl:text>
47  \par </xsl:text>
48  <xsl:apply-templates/>
49  </xsl:template>
50
51  <xsl:template match="emph">
52  <xsl:text>\emph{</xsl:text>
53  <xsl:apply-templates/>
54  <xsl:text>}</xsl:text>
55  </xsl:template>
56
57  <xsl:template match="back">
58  <xsl:text>\begin{Back}
59  \Signature{</xsl:text>
60  <xsl:value-of select="signature"/>
61  <xsl:text>}
62  \end{Back}
63  </xsl:text>
64  </xsl:template>
65
66  </xsl:stylesheet>
```

La ligne 2 définit que ce qui suit est une feuille de style XSL qui correspond à la première version de la spécification et utilise le domaine nominal représenté par `xsl` qui correspond à l'URI `http://www.w3.org/1999/XSL/Transform`. Si un autre URI est spécifié, la feuille de style ne pourra être traitée.

La ligne 4 indique que nous ne désirons pas générer un fichier XML (par défaut) mais un fichier « texte » (`method="text"`).

Pour traiter l'information présente dans un document il existe essentiellement deux approches :

1. « extraction individuelle » (méthode *pull*) avec sélection de la valeur des éléments à l'aide d'instructions XSLT `xsl:value-of`.
2. « distribution sélective » (méthode (*push*) à l'intérieur des motifs de sélection à l'aide d'instructions `xsl:apply-templates`.

En règle générale, ces deux méthodes sont utilisées dans une même feuille de style. Ainsi, ligne 13, nous « poussons » les éléments contenus à l'intérieur de l'élément `invitation` (sélectionné par le motif en ligne 8) à travers la feuille de style par l'instruction `<xsl:apply-templates/>`. Nous faisons la même opération à ligne 39 pour l'élément `body`, en ligne 48 pour les éléments `par` et en ligne 53 pour les éléments `emph`.

Par contre, pour le contenu de l'élément `front` (ligne 18), nous allons « tirer » les données présentes dans les différents éléments en nous servant des instructions de type

`<xsl:value-of .../>`. Ainsi, ligne 21, nous obtenons la « valeur » (c'est-à-dire : le contenu textuel dans ce cas) de l'élément `to` qui est un enfant de `front`, et ainsi de suite pour l'élément `date` (ligne 24), `where` (ligne 27) et `why` (ligne 30). Similairement pour l'élément `back`, qui est sélectionné par le motif en ligne 57, nous obtenons la valeur de l'élément `signature` en ligne 60.

Les autres composantes intéressantes de cette feuille de style sont les commandes LaTeX qui sont introduites à l'aide d'instructions `<xsl:text>` pour chacune des balises XML. Ainsi pour l'élément racine `invitation` nous écrivons le préambule du document LaTeX (lignes 9–11) puis, à la fin (après avoir traité le reste du document grâce à l'instruction XSL `<xsl:apply-templates/>` de la ligne 13), nous terminons le fichier LaTeX avec `\end{document}` (ligne 14).

Remarquons que chaque balise XML a pratiquement un équivalent en LaTeX. C'est le fichier d'extension LaTeX `invitation.sty` (ligne 10) qui interprétera ces balises, comme on peut l'observer dans le fichier source `invitation.sty` suivant :

```
1  % invitation.sty
2  % Package to format invitation.xml
3  \setlength{\parskip}{1ex}
4  \setlength{\parindent}{0pt}
5  \pagestyle{empty}%% Turn off page numbering
6  \RequirePackage{array}
7  \newenvironment{Front}%
8    {\begin{center}\huge \sffamily Memorandum\end{center}
9     \begin{flushleft}
10    \begin{tabular}{@{}>{\bfseries}p{.2\linewidth}@{}p{.8\linewidth}@{}}\hline
11   }
12   {To whom:  & \@To     \\
13    Occasion: & \@Why    \\
14    Venue:    & \@Where  \\
15    When:     & \@Date   \\\hline
16    \end{tabular}
17    \end{flushleft}
18   }
19 \newenvironment{Body}{\vspace*{\parskip}}{\vspace*{\parskip}}
20 \newenvironment{Back}
21   {\begin{flushleft}}
22   {\hspace*{.5\linewidth}\fbox{\emph{\@Sig}}
23    \end{flushleft}
24   }
25 \newcommand{\To}[1]{\gdef\@To{#1}}
26 \newcommand{\Date}[1]{\gdef\@Date{#1}}
27 \newcommand{\Where}[1]{\gdef\@Where{#1}}
28 \newcommand{\Why}[1]{\gdef\@Why{#1}}
29 \newcommand{\Signature}[1]{\gdef\@Sig{#1}}
30 \endinput
```

Nous avons opté pour une correspondance générique entre les types d'élément XML et leur représentation LaTeX. Nous n'avons pas traduit ces éléments directement en constructions élémentaires mais nous avons utilisé des commandes et environnements spécifiques qui « mappent » les noms dans le document source sur un nom LaTeX (ainsi l'élément `<to>` est « mappé » sur la commande `\To`, etc.). Cette approche

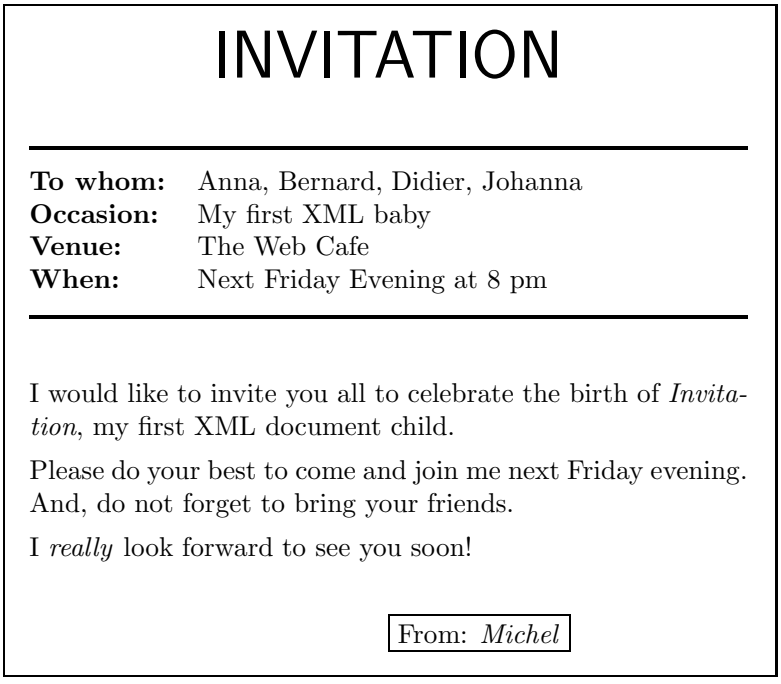


FIGURE 3 – *Résultat du formatage d'un document* XML *avec* LaTeX

nous permet de généraliser cette extension à d'autres langues (en particulier le français, voir page 41).

La seule chose qui nous manque pour transformer le XML en un autre format est un programme d'analyse qui réalise les recommandations XML et XSL. Il en existe plusieurs, mais dans ce qui suit nous en utilisons uniquement deux, dont le programme `xt`[5] de James CLARK, un des éditeurs des recommandations de la famille XSL. Il s'agit d'une bibliothèque Java de type `jar` qui permet de transformer un document XML dans un autre format d'après les règles spécifiées dans une feuille de style XSL.

La commande suivante montre comment utiliser le processeur `xt`. Nous spécifions le fichier source XML `invitation.xml`, la feuille de style XSL `invlat1.xsl` à l'entrée et nous obtenons le fichier `invlat1.tex` à la sortie.

```
java com.jclark.xsl.sax.Driver invitation.xml invlat1.xsl invlat1.tex
```

Après compilation du fichier TeX `invlat1.tex` avec LaTeX en utilisant l'extension `invitation.sty` nous obtenons le résultat montré à la figure 3.

6.1.2. Sortie HTML

Nous savons déjà que nous pouvons baliser un document XML de plusieurs façons. Dans la première version que nous venons d'étudier, nous avons privilégié l'utilisation de types d'élément distincts pour les différentes parties structurelles de l'information. Une autre possibilité (que nous explorons ci-dessous) spécifie une partie de l'information à l'aide d'attributs, ce qui minimise le nombre de balises. Voici la DTD `invitation2.dtd` qui définit la structure du balisage pour cette deuxième version du fichier XML :

```
<!-- invitation2 DTD     -->
<!ENTITY % i18n   "xml:lang NMTOKEN #IMPLIED">
<!ELEMENT invitation (par+)>
<!ATTLIST invitation  %i18n;
                      date      CDATA #REQUIRED
                      to        CDATA #REQUIRED
                      signature CDATA #REQUIRED
                      where     CDATA #REQUIRED
                      why       CDATA #IMPLIED
>
<!ELEMENT par        (#PCDATA|emph)*>
<!ELEMENT emph       (#PCDATA)>
```

La DTD ne contient plus que trois éléments : `invitation` (la racine), `par` et `emph`. D'après cette DTD nous avons saisi le fichier source XML `invitation2.xml` suivant :

```
<?xml version="1.0"?>
<!DOCTYPE invitation SYSTEM "invitation2.dtd">
<invitation to="Anna, Bernard, Didier, Johanna"
            date="Next Friday Evening at 8 pm"
            where="The Web Cafe"
            why="My first XML baby"
            signature="Michel"
>
<par>
I would like to invite you all to celebrate
the birth of <emph>Invitation</emph>, my
first XML document child.
</par>
<par>
Please do your best to come and join me next Friday
evening. And, do not forget to bring your friends.
</par>
<par>
I <emph>really</emph> look forward to see you soon!
</par>
</invitation>
```

Ce fichier `invitation2.xml` contient effectivement la même information que le fichier `invitation.xml` de la section précédente. En particulier toutes les informations génériques concernant l'invitation (`date`, `to`, etc.) sont devenues des attributs de l'élément racine `invitation`.

Pour transformer le fichier XML `invitation2.xml` en HTML nous utilisons le fichier de style XSL `invhtml2.xsl` suivant :

```
1  <?xml version='1.0'?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3
4  <xsl:output method="html"/>
5  <xsl:preserve-space elements="*"/>
6
7  <xsl:template match="invitation">
8  <html>
9  <head>
10 <title> Invitation (XSL/CSS formatting) </title>
11 <link href="invit.css" rel="stylesheet" type="text/css"/>
12 <!-- 12 November 1998 mg -->
13 </head>
14 <body>
15 <h1>INVITATION</h1>
16 <table>
17 <tbody>
18 <tr><td class="front">To: </td>
19 <td><xsl:value-of select="@to"/></td></tr>
20 <tr><td class="front">When: </td>
21 <td><xsl:value-of select="@date"/></td></tr>
22 <tr><td class="front">Venue: </td>
23 <td><xsl:value-of select="@where"/></td></tr>
24 <tr><td class="front">Occasion: </td>
25 <td><xsl:value-of select="@why"/></td></tr>
26 </tbody>
27 </table>
28 <xsl:apply-templates/>
29 <p class="signature"><xsl:value-of select="@signature"/></p>
30 </body>
31 </html>
32 </xsl:template>
33
34 <xsl:template match="par">
35 <p><xsl:apply-templates/></p>
36 </xsl:template>
37
38 <xsl:template match="emph">
39 <em><xsl:apply-templates/></em>
40 </xsl:template>
41
42 </xsl:stylesheet>
```

La ligne 4 déclare (implicitement) que le domaine nominal par défaut est HTML, ce qui veut dire que chaque balise non qualifiée est considérée comme du HTML. Les motifs des lignes 7, 34 et 42 traitent les trois éléments utilisés dans notre document XML. L'information concernant les destinataires (attribut `to`), la date (attribut `date`), etc., est extraite avec des instructions `<xsl:value-of .../>` (lignes 19, 21, etc.). Nous avons utilisé l'élément HTML `h1` (ligne 15) pour afficher le titre, alors que l'entête est générée à l'aide d'une table (lignes 16–27) et l'information adéquate (mot-clé et la valeur correspondante) y est introduite dans les cellules, rangée par rangée. Signalons également que nous utilisons l'attribut `class` sur plusieurs balises HTML (lignes 19, 21, . . . , 29) pour mieux piloter la représentation dans un butineur à l'aide

d'une feuille de style CSS [10, 39, 40] qui permet de contrôler les paramètres de visualisation. La ligne 11 déclare comment associer le fichier HTML avec le fichier CSS `invit.css` suivant :

```
1  /* CSS stylesheet for invitation1 in HTML */
2  BODY {margin-top: 1em;     /* global page parameters */
3        margin-bottom: 1em;
4        margin-left: 1em;
5        margin-right: 1em;
6        font-family: serif;
7        line-height: 1.1;
8        color: black;
9  }
10 H1   {text-align: center;  /* for global title   */
11       font-size: x-large;
12 }
13 P    {text-align: justify; /* paragraphs in body */
14       margin-top: 1em;
15 }
16 TABLE { border-width: 0pt }
17 TBODY { border-width: 0pt }
18 TD[class="front"] {        /* table data in front matter */
19       text-align: left;
20       font-weight: bold;
21 }
22 TD.front {        /* table data in front matter */
23       text-align: left;
24       font-weight: bold;
25 }
26 EM   {font-style: italic;  /* emphasis in body   */
27 }
28 P.signature {     /* signature          */
29       text-align: right;
30       font-weight: bold;
31 }
```

Nous obtenons le fichier HTML `invhtml2.html` avec le programme `xt` avec la commande suivante :

```
xt invitation2.xml invhtml2.xsl invhtml2.html
```

La figure 4 montre le résultat HTML visualisé à l'aide du butineur Microsoft Explorer en utilisant la feuille de style CSS `invit.css` montré ci-dessus.

6.1.3. Générer des objets de formatage XSL

Un fichier de style `invfo1.xsl` qui traduit le fichier source XML `invitation.xml` en des objets de formatage XSL est le suivant :

```
1  <?xml version='1.0'?>
2  <xsl:stylesheet version="1.0"
3                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

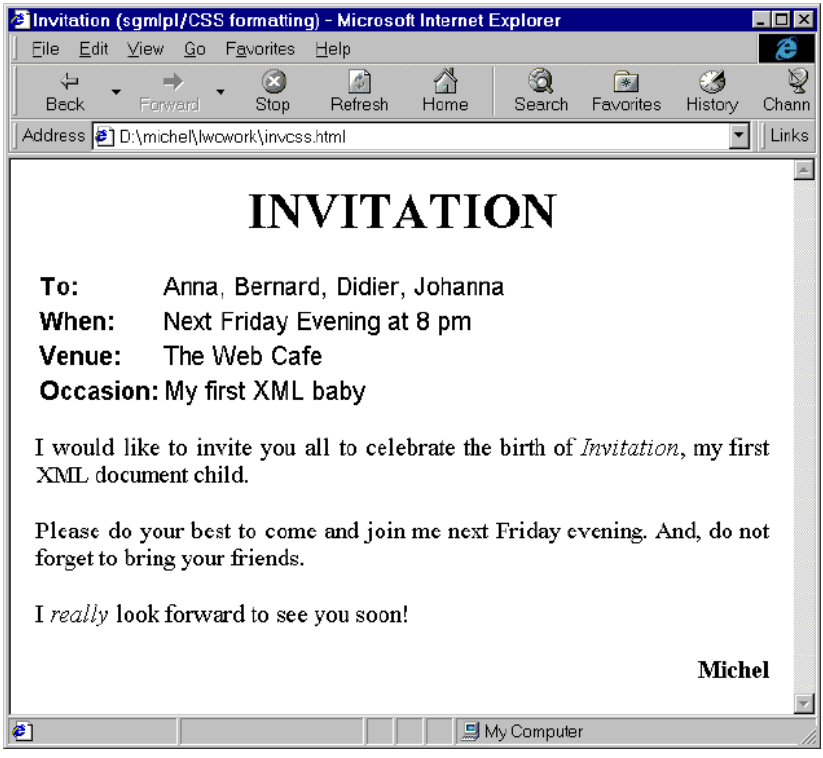


FIGURE 4 – *Le fichier* HTML `invhtml2.html` *visualisé par MS Explorer*

```
4                 xmlns:fo="http://www.w3.org/1999/XSL/Format">
5
6   <xsl:strip-space elements="*"/>
7
8   <!-- Paramétrisations -->
9
10  <xsl:variable name="PageMarginTop">75pt</xsl:variable>
11  <xsl:variable name="PageMarginBottom">125pt</xsl:variable>
12  <xsl:variable name="PageMarginLeft">80pt</xsl:variable>
13  <xsl:variable name="PageMarginRight">150pt</xsl:variable>
14  <xsl:variable name="BodySize">12pt</xsl:variable>
15
16  <xsl:template name="listitem">
17    <xsl:param name="labeltext">labeltext</xsl:param>
18    <xsl:param name="itemid">itemid</xsl:param>
19    <xsl:param name="itemtext">itemtext</xsl:param>
20    <fo:list-item id="{$itemid}">
21      <fo:list-item-label font-style="italic">
22        <fo:block>
23          <xsl:value-of select="$labeltext"/>
24          <xsl:text>:</xsl:text>
25        </fo:block>
26      </fo:list-item-label>
```

```
    <fo:list-item-body>
      <fo:block><xsl:value-of select="$itemtext"/></fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>

<xsl:template match='/'>
  <fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">
    <fo:layout-master-set>
      <fo:simple-page-master
        page-master-name="allpages"
        margin-top="{$PageMarginTop}"
        margin-bottom="{$PageMarginBottom}"
        margin-left="{$PageMarginLeft}"
        margin-right="{$PageMarginRight}">
        <fo:region-body margin-bottom="100pt"/>
        <fo:region-after extent="25pt"/>
      </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence>
        <fo:sequence-specification>
          <fo:sequence-specifier-repeating
            page-master-first="allpages"
            page-master-repeating="allpages"/>
        </fo:sequence-specification>
        <fo:flow font-family="serif">
          <xsl:apply-templates/>
        </fo:flow>
      </fo:page-sequence>
  </fo:root>
</xsl:template>

<xsl:template match="front">
<fo:block font-family="sans-serif"  font-size="24pt"
          font-weight="bold"        text-align-last="centered"
          space-after.optimum="24pt">
  <xsl:text>INVITATION</xsl:text>
</fo:block>

<fo:list-block provisional-distance-between-starts="2cm"
               provisional-label-separation="6pt">
  <xsl:call-template name="listitem">
    <xsl:with-param name="labeltext">To</xsl:with-param>
    <xsl:with-param name="itemid">listto</xsl:with-param>
    <xsl:with-param name="itemtext"><xsl:value-of select="to"/></xsl:with-param>
  </xsl:call-template>
  <xsl:call-template name="listitem">
    <xsl:with-param name="labeltext">When</xsl:with-param>
    <xsl:with-param name="itemid">listdate</xsl:with-param>
    <xsl:with-param name="itemtext"><xsl:value-of select="date"/></xsl:with-param>
  </xsl:call-template>
  <xsl:call-template name="listitem">
    <xsl:with-param name="labeltext">Venue</xsl:with-param>
    <xsl:with-param name="itemid">listwhere</xsl:with-param>
    <xsl:with-param name="itemtext" select="where"/>
  </xsl:call-template>
  <xsl:call-template name="listitem">
    <xsl:with-param name="labeltext">Occasion</xsl:with-param>
    <xsl:with-param name="itemid">listwhy</xsl:with-param>
    <xsl:with-param name="itemtext"><xsl:value-of select="why"/></xsl:with-param>
  </xsl:call-template>
</fo:list-block>
</xsl:template>
```

```
90
91  <xsl:template match="par">
92    <fo:block space-before.optimum="{$BodySize}">
93      <xsl:apply-templates/>
94    </fo:block>
95  </xsl:template>
96
97  <xsl:template match="emph">
98    <fo:inline-sequence font-style="italic">
99      <xsl:apply-templates/>
100   </fo:inline-sequence>
101 </xsl:template>
102
103 <xsl:template match="back">
104 <fo:block space-before.optimum="{$BodySize}"
105           font-weight="bold" text-align-last="end">
106    <xsl:text>From: </xsl:text>
107    <xsl:value-of select="signature"/>
108 </fo:block>
109 </xsl:template>
110
111 </xsl:stylesheet>
```

Nous n'essaierons pas d'expliquer en détail toutes les lignes de cette feuille de style (la norme XSLF [52] n'est d'ailleurs pas encore terminée). Nous mentionnons toutefois en ligne 4 la définition du domaine nominal `fo`, utilisé partout dans le fichier.

Des variables XSL sont définies en lignes 10–14 et utilisées, par exemple, en lignes 38–41.

La définition du « modèle nommé » (*named template*) `listitem` se trouve en lignes 16–31. Nous y voyons comment les trois paramètres `labeltext` (ligne 17), `itemid` (ligne 18) et `itemtext` (ligne 19) sont déclarés et comment une valeur par défaut leur est associée. Les valeurs de ces paramètres sont utilisées en lignes 23, 20 et 28, respectivement (la valeur d'un paramètre ou d'une variable est utilisée en faisant précéder son nom par un dollar `$` et en entourant le tout d'accolades, comme en ligne 20 avec `id="{$itemid}"`). Leurs valeurs peuvent être définies quand le modèle nommé `listitem` est invoqué (lignes 68–72, 73–77, 78–82 et 83–87).

En traitant le fichier source XML `invitation.xml` et la feuille de style `invfo1.xsl` avec le processeur XSLT `xt`, nous obtenons un fichier `invfo1.fo`, qui contient une représentation basée sur des objets de formatage XSL.

```
xt invitation.xml invfo1.xsl invfo1.fo
```

Pour compiler le fichier `invfo1.fo` nous disposons de deux programmes librement accessibles : FOP [34] de James TAUBER et PassiveTEX[29, 30] de Sebastian RAHTZ. Le premier traduit les objets directement en PDF, alors que PassiveTEX utilise `pdftex` (donc le moteur formatage de TEX) pour produire le PDF. La figure 5 montre le résultat obtenu par ces deux approches.

$$\text{XML} \xrightarrow{\text{XSL}} \text{FO} \xrightarrow{\text{FOP}} \text{PDF}$$

INVITATION

To: Anna, Bernard, Didier, Johanna
When: Next Friday Evening at 8 pm
Venue: The Web Cafe
Occasion: My first XML baby

I would like to invite you all to celebrate the birth of *Invitation* , my first XML document child.

Please do your best to come and join me next Friday evening. And, do not forget to bring your friends.

I *really* look forward to see you soon!

From: Michel

$$\text{XML} \xrightarrow{\text{XSL}} \text{FO} \xrightarrow{\text{PassiveTeX}} \text{PDF}$$

INVITATION

To: Anna, Bernard, Didier, Johanna
When: Next Friday Evening at 8 pm
Venue: The Web Cafe
Occasion: My first XML baby

I would like to invite you all to celebrate the birth of *Invitation*, my first XML document child.

Please do your best to come and join me next Friday evening. And, do not forget to bring your friends.

I *really* look forward to see you soon!

From: Michel

FIGURE 5 – *Résultat* PDF *obtenu avec FOP et avec PassiveTEX*

6.1.4. Exemple en français

Jusqu'ici nous avons étudié des exemples XML d'une invitation en anglais. Toutefois, pour une utilisation par des francophones, il est plus commode de baliser le document en français. Un exemple de noms de balises et d'un texte en français se trouve dans le fichier `invitationfr.xml` suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE invitation SYSTEM "invitationfr.dtd">
<invitation>
<!-- ++++ Partie entête ++++ -->
<entête>
<à>Anna, Bernard, Didier, Johanna</à>
<date>Vendredi prochain à 20 heures</date>
<où>Le Café du Web</où>
<pourquoi>Mon premier bébé XML</pourquoi>
</entête>
<!-- ++++ Partie corps  ++++ -->
<corps>
<par>
J'ai le plaisir de vous inviter à la célébration
de la naissance d'<emph>Invitation</emph>, mon
premier enfant document XML.
</par>
<par>
S'il vous plaît, faites tout votre possible pour me rejoindre
vendredi prochain. Et n'oubliez pas d'emmener vos amis.
</par>
<par>
Je me réjouis <emph>vraiment</emph> d'avance de votre présence.
</par>
</corps>
<!-- ++++ Partie finale  ++++ -->
<fin>
<signature>Michel</signature>
</fin>
</invitation>
```

La version française de la DTD a la même architecture que son analogue anglais, tout en utilisant des noms francisés. Le contenu du fichier DTD `invitationfr.dtd` suit :

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<!-- DTD invitation (version française) -->
<!-- 11 novembre 1998 mg -->
<!ELEMENT invitation (entête, corps, fin) >
<!ELEMENT entête     (à, date, où, pourquoi?) >
<!ELEMENT date       (#PCDATA) >
<!ELEMENT à          (#PCDATA) >
<!ELEMENT où         (#PCDATA) >
<!ELEMENT pourquoi   (#PCDATA) >
<!ELEMENT corps      (par+) >
<!ELEMENT par        (#PCDATA|emph)* >
<!ELEMENT emph       (#PCDATA) >
<!ELEMENT fin        (signature) >
<!ELEMENT signature  (#PCDATA) >
```

Notons l'utilisation de caractères accentués pour certains noms d'éléments, comme `entête`, à et où. Le fichier d'extensions LaTeX `invitationfr.sty`, montré ci-dessous, contient les définitions des commandes et environnements qui correspondent aux différentes balises du fichier XML (versions anglaise et française).

```
1  % invitation.sty
2  % Package to format invitation.xml
3  \setlength{\textwidth}{22pc}
4  \setlength{\parskip}{1ex}
5  \setlength{\parindent}{0pt}
6  \pagestyle{empty}%% Turn off page numbering
7  \RequirePackage{array,calc}
8  \newcommand{\ToTitle}{To whom}
9  \newcommand{\WhyTitle}{Occasion}
10 \newcommand{\WhereTitle}{Venue}
11 \newcommand{\DateTitle}{When}
12 \newcommand{\SignatureTitle}{From}
13 \DeclareOption{francais}{%  French text for fixed texts
14   \renewcommand{\ToTitle}{À}
15   \renewcommand{\WhyTitle}{À l'occasion de}
16   \renewcommand{\WhereTitle}{Où}
17   \renewcommand{\DateTitle}{Quand}
18   \renewcommand{\SignatureTitle}{De la part de}}
19 \newenvironment{Front}%
20   {\begin{center}
21      \Huge\sffamily INVITATION
22    \end{center}
23   }
24   {\begin{flushleft}
25    \rule{\linewidth}{1pt}\\[2mm]
26    \begin{tabular}{@{}>{\bfseries}ll@{}}
27    \ToTitle:    & \@To     \\
28    \WhyTitle:   & \@Why    \\
29    \WhereTitle: & \@Where  \\
30    \DateTitle:  & \@Date
31    \end{tabular}\\[2mm]
32    \rule{\linewidth}{1pt}
33    \end{flushleft}
34   }
35 \newenvironment{Body}{\vspace*{\parskip}}{\vspace*{\parskip}}
36 \newenvironment{Back}
37   {\begin{flushleft}}
38   {\hspace*{.5\linewidth}\fbox{\SignatureTitle: \emph{\@Sig}}
39    \end{flushleft}
40   }
41 \newcommand{\To}[1]{\gdef\@To{#1}}
42 \newcommand{\Date}[1]{\gdef\@Date{#1}}
43 \newcommand{\Where}[1]{\gdef\@Where{#1}}
44 \newcommand{\Why}[1]{\gdef\@Why{#1}}
45 \newcommand{\Signature}[1]{\gdef\@Sig{#1}}
46
47 \ProcessOptions
48
49 \endinput
```

Il s'agit d'une généralisation du fichier `invitation.sty` introduit à la section 6.1.1. Nous y avons introduit une paramétrisation des textes imprimés pour les différentes entrées dans l'en-tête, avec une dépendance de l'option `francais` (lignes 13–18, une généralisation à d'autres langues serait triviale).

Le fichier XSL `invlat1fr.xsl` pour traduire l'invitation en français en LaTeX suit :

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

<xsl:output method="text" indent="no" encoding="ISO-8859-1"/>

<xsl:strip-space elements="*"/>

<xsl:template match="/">
<xsl:text>\documentclass[francais]{article}
\usepackage{invitationfr}
\usepackage[T1]{fontenc}
\begin{document}
</xsl:text>
<xsl:apply-templates/>
<xsl:text>\end{document}
</xsl:text>
</xsl:template>

<xsl:template match="entête">
<xsl:text>\begin{Front}
\To{</xsl:text>
<xsl:value-of select="à"/>
<xsl:text>}
\Date{</xsl:text>
<xsl:value-of select="date"/>
<xsl:text>}
\Where{</xsl:text>
<xsl:value-of select="où"/>
<xsl:text>}
\Why{</xsl:text>
<xsl:value-of select="pourquoi"/>
<xsl:text>}
\end{Front}
</xsl:text>
</xsl:template>

<xsl:template match="corps">
<xsl:text>\begin{Body}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>\end{Body}
</xsl:text>
</xsl:template>

<xsl:template match="par">
<xsl:text>\par</xsl:text>
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="emph">
<xsl:text>\emph{</xsl:text>
<xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>

<xsl:template match="fin">
<xsl:text>\begin{Back}
\Signature{</xsl:text>
```

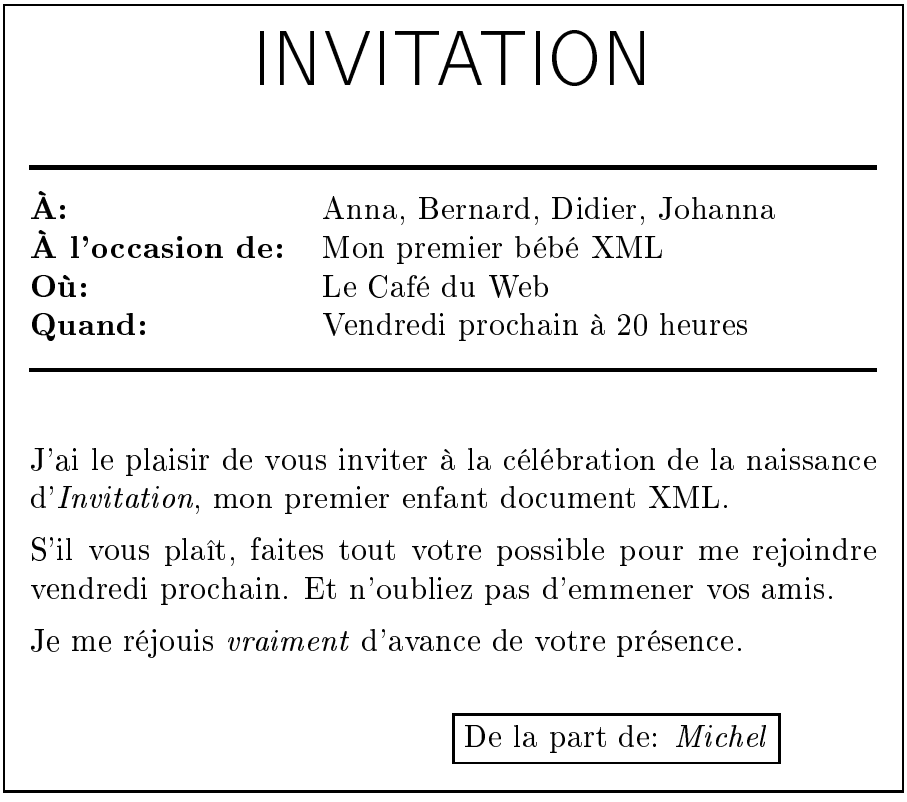


FIGURE 6 – *Résultat du formatage d'une invitation française avec LaTeX*

```
61  <xsl:value-of select="signature"/>
62  <xsl:text>}
63  \end{Back}
64  </xsl:text>
65  </xsl:template>
66
67  </xsl:stylesheet>
```

À l'exception du remplacement des noms d'élément, par leur version française, cette version de la feuille de style est tout à fait identique au fichier `invlat1.xsl` décrit à la section 6.1.1.

Après la génération avec `xt` d'un fichier LaTeX `invlat1fr.tex` à partir du fichier source `invitationfr.xml` et de la feuille de style XSL `invlat1fr.xsl`, nous obtenons la présentation de la figure 6.

6.1.5. Alphabets non-latins et documents codés en UTF-8

Nous avons souligné, dans la discussion du langage XML, que celui-ci est véritablement international car basé sur Unicode. Pour voir comment nous pouvons exploiter optimalement cette possibilité nous avons saisi, à l'aide d'un éditeur qui permet l'utilisation du codage UTF-8, un document qui contient du grec, du cyrillique et quelques autres symboles Unicode. Le document source `utf8.xml` est montré en figure 7 à l'intérieur d'une fenêtre de l'éditeur `yudit`[32].

Nous reprenons ci-dessous les premières lignes de ce fichier XML.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE mydoc [
3  <!ELEMENT mydoc (#PCDATA)>
4  <!ENTITY % ISOcyr1 SYSTEM "ISOcyr1.pen">
5  %ISOcyr1;
6  ]>
7  <mydoc>
8  <par>The word Russian (ĐăÑČŇĄÑĄĐžĐÿĐź) in Cyrillic: <br/>
9  Using ISO Cyrillic set:
10 &Rcy;&ucy;&scy;&scy;&kcy;&icy;&jcy;  <br/>
11 Using XML Unicode entities:
12 &#x0420;&#x0443;&#x0441;&#x0441;&#x043a;&#x0438;&#x0439;
13 </par>
```

La ligne 4 définit l'entité paramètre `ISOcyr1` qui l'associe au fichier `ISOcyr1.pen`. La ligne 5 est un appel d'entité paramètre, qui indique que toutes les déclarations présentes dans ce fichier sont inclues à cet endroit dans le sous-ensemble interne. Le fichier `ISOcyr1.pen` contient par exemple les déclarations suivantes :

```
1  <!ENTITY rcy    "&#x440;"> <!--small er, Cyrillic -->
2  <!ENTITY Rcy    "&#x420;"> <!--capital ER, Cyrillic -->
3  <!ENTITY scy    "&#x441;"> <!--small es, Cyrillic -->
4  <!ENTITY Scy    "&#x421;"> <!--capital ES, Cyrillic -->
```

Ces entités sont référencées en ligne 10 du document source, alors qu'en ligne 12 nous utilisons une représentation hexadécimale équivalente pour ces lettres cyrilliques. La ligne 8 en présente une forme en UTF-8, qui est indéchiffrable telle que, mais qui est lisible en figure 7.

La figure 8 montre la feuille de style XSL qui transforme le fichier source `utf8.xml` en HTML. À la première ligne, nous déclarons que le fichier de sortie est en HTML et qu'il sera codé en UTF-8. La fin du fichier montre comment nous pouvons sélectionner le texte russe en utilisant une représentation Unicode en appel de caractères ou directement en code UTF-8, comme la puce (•) ou le motif grec ($\varepsilon\lambda\lambda$).

Le résultat final de la transformation du fichier source XML de la figure 7 avec la feuille de style XSL de la figure 8 en HTML visualisé avec le butineur Netscape est montré en figure 9.

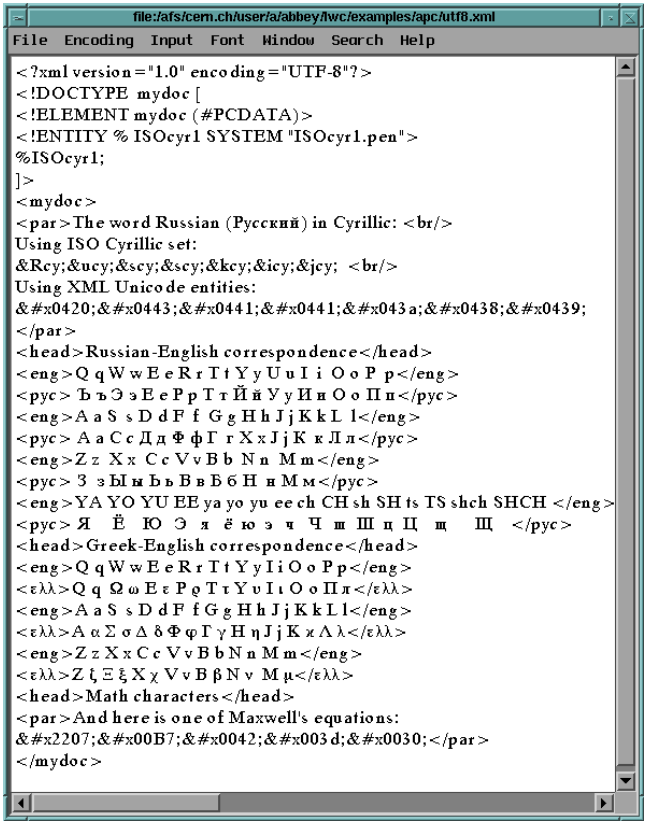


FIGURE 7 – *Fichier source* XML `utf8.xml` *avec du grec, du russe et des mathématiques (codage* UTF–8*)*

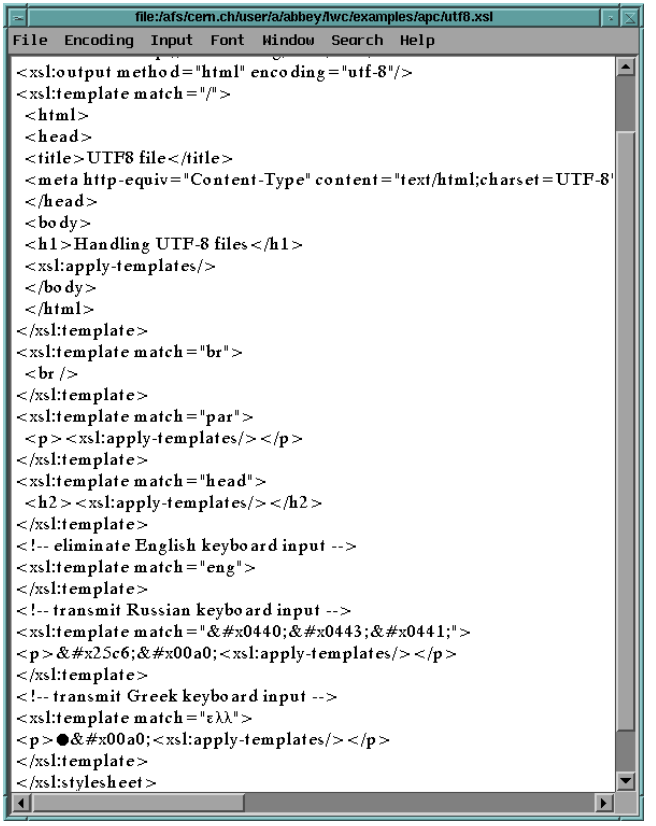


FIGURE 8 – *Fichier source* XSL `utf8.xsl` *pour transformer le fichier de la figure 7 en* HTML

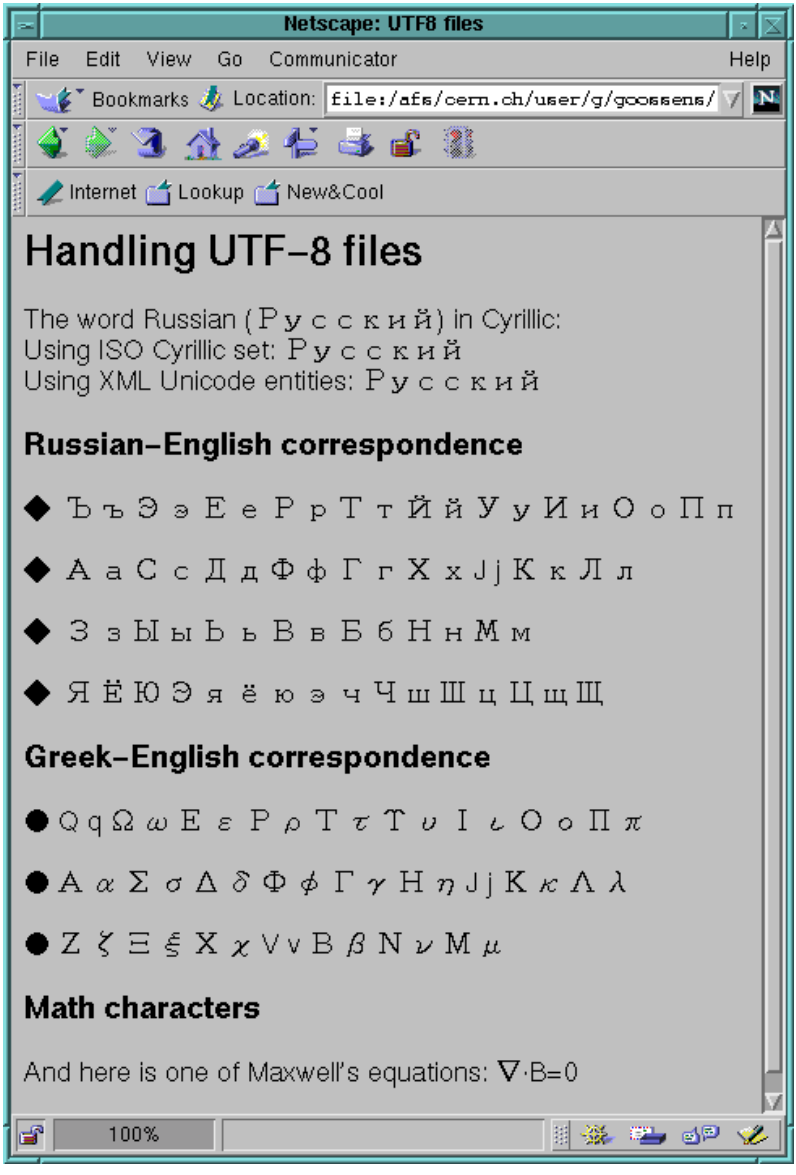


FIGURE 9 – *Résultat avec* HTML *d'un fichier UTF8 avec Netscape*

6.2. XML et XSL pour manipuler une base de données

Dans notre discussion de XSL comme langage de style, nous nous sommes intéressés jusqu'ici principalement aux aspects de formatage de l'information dans le but de la visualiser.

Toutefois XSLT, la partie transformation de XSL, peut aussi servir comme base pour manipuler (au sens large) des données[3]. Il n'est pas notre intention de reproduire dans ces pages les nombreuses discussions sur la définition d'un langage d'interrogation (*query language*) pour des bases de données XML (un atelier de deux jours a rassemblé une centaine de participants à Boston en décembre 1998 où plus de soixante propositions ont été discutées pour définir un tel langage [42]).

6.2.1. Préparation d'un fichier XML

Pour montrer comment utiliser XSLT avec une base de données, nous avons choisi de partir de la série de tables contenant les noms des pays et de leurs monnaies en plusieurs langues préparée par les services de l'Union européenne (voir l'URL `http://europa.eu.int/comm/sdt/currencies/frtable1.htm` pour la version française). Cette table est balisée en HTML et a la structure suivante :

```
1   <TABLE CELLPADDING="5" WIDTH="95%" BORDER="1">
2     <TR VALIGN="TOP" ALIGN="LEFT" BGCOLOR="#FFFF80">
3       <TD WIDTH="5%"><SMALL><B>Code ISO</B><A HREF="#fn2"
4         ONMOUSEOVER="window.status='norme ISO 3166 ... '; return true"
5         ONMOUSEOUT="window.status=''; return true"><SUP>2</SUP></A></SMALL></TD>
6       <TD WIDTH="11%"><SMALL><B>Forme courte</B></SMALL></TD>
7       <TD WIDTH="16%"><SMALL><B>Forme longue</B></SMALL></TD>
8       <TD WIDTH="11%"><SMALL><B>Capitale</B></SMALL></TD>
9       <TD WIDTH="11%"><SMALL><B>Citoyen ou habitant</B></SMALL></TD>
10      <TD WIDTH="14%"><SMALL><B>Adjectif</B></SMALL></TD>
11      <TD WIDTH="16%"><SMALL><B>Unit&eacute; mon&eacute;taire</B></SMALL></TD>
12      <TD WIDTH="5%"><SMALL><B>Code ISO</B><A HREF="#fn3"
13        ONMOUSEOVER="window.status='norme ISO 4217 ...'; return true"
14        ONMOUSEOUT="window.status=''; return true"><SUP>3</SUP></A></SMALL></TD>
15      <TD WIDTH="11%"><SMALL><B>Subdivision de l'unit&eacute;
16                            mon&eacute;taire</B></SMALL></TD>
17    </TR>
18    <TR VALIGN="TOP">
19      <TD><SMALL><B>AD</B></SMALL></TD>
20      <TD><SMALL>Andorre</SMALL></TD>
21      <TD><SMALL>la Principaut&eacute; d'Andorre</SMALL></TD>
22      <TD><SMALL>Andorre-la-Vieille</SMALL></TD>
23      <TD><SMALL>Andorran</SMALL></TD>
24      <TD><SMALL>andorran</SMALL></TD>
25      <TD><SMALL>peseta; franc fran&ccedil;ais</SMALL></TD>
26      <TD><SMALL><B>ESP; FRF</B></SMALL></TD>
27      <TD><SMALL>centimo; centime</SMALL></TD>
28    </TR>
```

D'abord, pour obtenir un fichier XML qui peut être manipulé avec les applications décrites dans cet article, nous avons traité le fichier HTML ci-dessus avec l'utilitaire `tidy`[28] qui « normalise » son contenu et le transforme en XML :

```
tidy -asxml -latin1 <frtable.html >frtable.xhtml
```

Puis nous avons enlevé toute information non directement liée aux pays et à leurs monnaies en éliminant les embellissements et les notes explicatives. Pour cela nous avons écrit un fichier de style XSL `filter.xsl` qui filtre tout ce que nous considérons comme inutile.

```
1  <?xml version='1.0' encoding="ISO-8859-1"?>
2  <xsl:stylesheet version="1.0"
3                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4  <xsl:output method="xml" encoding="ISO-8859-1"/>
5  <xsl:strip-space elements="td"/>
6
7  <xsl:template match="*" priority="-2">
8  <xsl:copy>
9  <xsl:apply-templates/>
10 </xsl:copy>
11 </xsl:template>
12
13 <xsl:template match="sup|a|i">
14 <!-- ignorer contenu -->
15 </xsl:template>
16 <xsl:template match="small|b">
17 <xsl:apply-templates/>
18 </xsl:template>
19
20 </xsl:stylesheet>
```

Le motif de correspondance générique (ligne 7–11) s'applique à tous les éléments dans le document XML qui n'ont pas de motif explicite spécifié (d'où l'utilisation de la priorité négative). Tous ces éléments sont copiés (en éliminant leur attributs) par l'instruction `<xsl:copy>`. La deuxième partie de la feuille de style élimine le contenu des éléments `sup`, `a` et `i` (lignes 13–15) et ne retient que l'information à l'intérieur des éléments `small` et `b`, en éliminant leurs balises (lignes 16–18).

Nous utilisons le processeur XSL `LotusXSL`[18], un autre processeur de feuille de style XSL, pour générer le fichier `frisotab-in.xml`, comme montré ci-dessous.

```
java com.lotus.xsl.Process -in frtable.xhtml -xsl filter.xsl -out frisotab-in.xml
========== Parsing and preparing filter1.xsl ==========
Parsing and init of filter.xsl took 1559 milliseconds
========== Parsing f.xml ==========
Parse of frtable.xml took 563 milliseconds
============================
Transforming frtable.xml via Input XSL...
transform took 3860 milliseconds
Total time took 4452 milliseconds
XSLProcessor: done
```

Le fichier `frisotab-in.xml` est beaucoup plus simple que l'original HTML et il a une structure XML :

```
<html>
<head>
<title>Pays et Monnaies</title>
</head>
<body>
<table>
<tr>
<td>Code ISO</td>
<td>Forme courte</td>
<td>Forme longue</td>
<td>Capitale</td>
<td>Citoyen ou habitant</td>
<td>Adjectif</td>
<td>Unité monétaire</td>
<td>Code ISO</td>
<td>Subdivision de l'unité monétaire</td>
</tr>

<tr>
<td>AD</td>
<td>Andorre</td>
<td>la Principauté d'Andorre</td>
<td>Andorre-la-Vieille</td>
<td>Andorran</td>
<td>andorran</td>
<td>peseta; franc français</td>
<td>ESP; FRF</td>
<td>centimo; centime</td>
</tr>
```

Maintenant, nous voulons transformer cette table en une base de données où les neuf composantes de l'information (voir lignes 8–16) sont bien étiquetées. En fait nous nous proposons de baliser l'information de deux façons différentes, l'une en nous basant sur neuf éléments distincts et l'autre sur un élément unique mais avec neuf attributs (c'est analogue aux deux instances de l'invitation que nous avons étudiées antérieurement à la section 6.1).

6.2.2. Base de données basée sur les types d'élément

Prenons le fichier source `frisotab-in.xml` généré précédemment et transformons-le dans une forme exploitable plus facilement comme base de donnée. À cet effet, nous avons écrit la feuille de style `isotab1.xsl` qui suit.

```
<?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" encoding="ISO-8859-1"/>
<xsl:strip-space elements="*"/>

<xsl:template match="/html">
<xsl:element name="countries">
```

```
<xsl:text>
</xsl:text>
<xsl:apply-templates/>
</xsl:element>
</xsl:template>

<xsl:template match="*" priority="-2">
<xsl:element name="UNKNOWN/INCONNU">
<xsl:value-of select="name(.)"/>
<xsl:apply-templates/>
</xsl:element>
</xsl:template>

<xsl:template match="head|tr">
<!-- ignore content -->
</xsl:template>

<xsl:template match="body|table">
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="tr[position()>1]" priority="2">
<xsl:element name="country">
<xsl:text>&#xA;</xsl:text>
<xsl:apply-templates/>
</xsl:element>
<xsl:text>&#xA;</xsl:text>
</xsl:template>

<xsl:template match="td[1]">
<xsl:element name="shortname">
<xsl:apply-templates/>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:template>

<xsl:template match="td[2]">
<xsl:element name="fullname">
<xsl:apply-templates/>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:template>

<xsl:template match="td[3]">
<xsl:element name="isocountry">
<xsl:apply-templates/>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:template>

<xsl:template match="td[4]">
<xsl:element name="capital">
<xsl:apply-templates/>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:template>

<xsl:template match="td[5]">
<xsl:element name="citizen">
<xsl:apply-templates/>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:template>

<xsl:template match="td[6]">
<xsl:element name="adjective">
<xsl:apply-templates/>
</xsl:element><xsl:text>&#xA;</xsl:text>
```

```
72  </xsl:template>
73
74  <xsl:template match="td[7]">
75  <xsl:element name="currency">
76  <xsl:apply-templates/>
77  </xsl:element><xsl:text>&#xA;</xsl:text>
78  </xsl:template>
79
80  <xsl:template match="td[8]">
81  <xsl:element name="isocurrency">
82  <xsl:apply-templates/>
83  </xsl:element><xsl:text>&#xA;</xsl:text>
84  </xsl:template>
85
86  <xsl:template match="td[9]">
87  <xsl:element name="currensubunit">
88  <xsl:apply-templates/>
89  </xsl:element><xsl:text>&#xA;</xsl:text>
90  </xsl:template>
91
92  </xsl:stylesheet>
```

Nous y voyons que l'information est entourée de l'élément racine `countries` (ouvert et fermé en ligne 8 et 12, respectivement). Les lignes 15–19 sont un garde-fou contre l'information incorrecte (un nom d'élément non reconnu). Les lignes 21–23 spécifient que nous ignorons l'information contenue dans les éléments `head` et `tr` (en général), alors que les lignes 25–27 copient le contenu des éléments `body` et `table` (sans leurs balises) dans le fichier de sortie. Comme nous balisons explicitement le contenu de chaque colonne de la table, nous n'avons pas besoin de garder l'en-tête de la table (la première ligne `tr`, lignes 7–17 du fichier `frisotab-in.xml`), d'où la condition dans le motif de sélection en ligne 29 qui ne choisit que les lignes plus grandes que deux (les lignes `tr` sont par défaut ignorées d'après le motif en ligne 21, d'où la spécification d'une priorité égale à deux garantissant que ce modèle remplace celui par défaut pour le motif en question). Nous y ouvrons un élément `country` en ligne 30 et nous ajoutons des paires `<xsl:text>` `<xsl:text>` contenant un retour à la ligne pour améliorer la lisibilité du fichier de sortie. Tout le restant du fichier correspond aux différents motifs de sélection de la première jusqu'à la neuvième colonne des lignes de la table où le contenu de chaque colonne sera copié dans un élément dont le type est défini par son contenu.

Avec le processeur XSL `lotusxsl` nous générons le fichier cible `frisotab1.xml`.

```
java com.lotus.xsl.Process -in frisotab-in.xml -xsl isotab1.xsl -out frisotab1.xml
========= Parsing and preparing isotab1.xsl ==========
Parsing and init of isotab1.xsl took 1135 milliseconds
========= Parsing frisotab-in.xml ==========
Parse of frisotab-in.xml took 368 milliseconds
============================
Transforming frisotab-in.xml via Input XSL...
transform took 30245 milliseconds
Total time took 30646 milliseconds
XSLProcessor: done
```

Voici le début du fichier `frisotab1.xml` :

```
<countries>
 <country>
  <shortname>AD</shortname>
  <fullname>Andorre</fullname>
  <isocountry>la Principauté d'Andorre</isocountry>
  <capital>Andorre-la-Vieille</capital>
  <citizen>Andorran</citizen>
  <adjective>andorran</adjective>
  <currency>peseta; franc français</currency>
  <isocurrency>ESP; FRF</isocurrency>
  <currensubunit>centimo; centime</currensubunit>
 </country>
```

Nous avons un peu arrangé la représentation ci-dessus pour mieux mettre en évidence la structure du document. Chaque élément de type `country` possède neuf enfants : `shortname`, `fullname`, . . . , qui correspondent aux neuf colonnes dans la table originale.

En section 6.1.1, nous avons expliqué que deux méthodes sont disponibles pour traiter l'information dans un fichier XML, l'extraction individuelle (*pull*) à l'aide d'instructions `<xsl:value-of .../>` et la distribution sélective (*push*) à l'aide d'instructions `<apply-templates/>`. Ci-dessus nous avons opté pour la deuxième approche. Toutefois nous pouvons obtenir le même fichier XML en sortie en utilisant la première méthode, comme dans le fichier `isotab1-bis.xsl` qui suit :

```
<?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" encoding="ISO-8859-1"/>
<xsl:strip-space elements="*"/>

<xsl:template match="/html">
<xsl:element name="countries">
<xsl:text>
</xsl:text>
<xsl:apply-templates/>
</xsl:element>
</xsl:template>

<xsl:template match="*" priority="-2">
<xsl:element name="UNKNOWN/INCONNU">
<xsl:apply-templates/>
</xsl:element>
</xsl:template>

<xsl:template match="head|tr" priority="0">
<!-- ignore content -->
</xsl:template>

<xsl:template match="body|table">
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="table/tr[position()>1]" priority="2">
<xsl:element name="country">
<xsl:text>
```

```
31  </xsl:text>
32  <xsl:element name="shortname">
33  <xsl:value-of select="td[1]"/>
34  </xsl:element>
35  <xsl:element name="fullname">
36  <xsl:value-of select="td[2]"/>
37  </xsl:element>
38  <xsl:element name="isocountry">
39  <xsl:value-of select="td[3]"/>
40  </xsl:element>
41  <xsl:element name="capital">
42  <xsl:value-of select="td[4]"/>
43  </xsl:element>
44  <xsl:element name="citizen">
45  <xsl:value-of select="td[5]"/>
46  </xsl:element>
47  <xsl:element name="adjective">
48  <xsl:value-of select="td[6]"/>
49  </xsl:element>
50  <xsl:element name="currency">
51  <xsl:value-of select="td[7]"/>
52  </xsl:element>
53  <xsl:element name="isocurrency">
54  <xsl:value-of select="td[8]"/>
55  </xsl:element>
56  <xsl:element name="currensubunit">
57  <xsl:value-of select="td[9]"/>
58  </xsl:element>
59  <xsl:text>
60  </xsl:text>
61  </xsl:element>
62  <xsl:text>
63  </xsl:text>
64  </xsl:template>
65
66  </xsl:stylesheet>
```

```
java com.lotus.xsl.Process -in frisotab-in.xml -xsl isotab1-bis.xsl -out frisotab1-bis.xml
========= Parsing and preparing isotab1-bis.xsl ==========
Parsing and init of isotab1-bis.xsl took 1109 milliseconds
========= Parsing frisotab-in.xml ==========
Parse of frisotab-in.xml took 353 milliseconds
============================
Transforming frisotab-in.xml via Input XSL...
transform took 18043 milliseconds
Total time took 18427 milliseconds
XSLProcessor: done
```

Nous avons remplacé le traitement des lignes `tr` de la table HTML en lignes 29–39 de la feuille `isotab1.xsl`, qui utilisait `<apply-templates/>` (ligne 33) pour pousser les données XML à travers les autres motifs en lignes 41–93 par un seul modèle dans le fichier `isotab1-bis.xsl` (lignes 28–64) qui extrait l'information à mettre dans les éléments (voir lignes 33, 36, etc.) à l'aide d'instructions `<xsl:value-of .../>`.

6.2.3. Base de données basée sur les attributs

Une deuxième forme de notre base de données n'utilise que le seul type d'élément `country` pour baliser l'information en la contenant dans des attributs. La feuille de style XSL `isotab2.xsl` qui transforme le fichier source en cette représentation suit :

```
<?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" encoding="ISO-8859-1"/>
<xsl:strip-space elements="*"/>

<xsl:template match="/html">
<xsl:element name="countries">
<xsl:text>
</xsl:text>
<xsl:apply-templates/>
</xsl:element>
</xsl:template>

<xsl:template match="*" priority="-2">
<xsl:element name="UNKNOWN/INCONNU">
<xsl:apply-templates/>
</xsl:element>
</xsl:template>

<xsl:template match="head|tr" priority="0">
<!-- ignore content -->
</xsl:template>

<xsl:template match="body|table">
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="table/tr[position()>1]" priority="2">
<xsl:element name="country">
<xsl:attribute name="shortname">
<xsl:value-of select="td[1]"/>
</xsl:attribute>
<xsl:attribute name="fullname">
<xsl:value-of select="td[2]"/>
</xsl:attribute>
<xsl:attribute name="isocountry">
<xsl:value-of select="td[3]"/>
</xsl:attribute>
<xsl:attribute name="capital">
<xsl:value-of select="td[4]"/>
</xsl:attribute>
<xsl:attribute name="citizen">
<xsl:value-of select="td[5]"/>
</xsl:attribute>
<xsl:attribute name="adjective">
<xsl:value-of select="td[6]"/>
</xsl:attribute>
<xsl:attribute name="currency">
<xsl:value-of select="td[7]"/>
</xsl:attribute>
<xsl:attribute name="isocurrency">
<xsl:value-of select="td[8]"/>
</xsl:attribute>
<xsl:attribute name="currensubunit">
<xsl:value-of select="td[9]"/>
</xsl:attribute>
</xsl:element>
<xsl:text>
</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Nous voyons que cette feuille de style est assez semblable à `isotab1-bis.xsl` qui génère la base de données en utilisant les types d'élément pour saisir l'information. La seule différence est l'emploi de l'instruction `<xsl:attribute name="...">` qui crée un attribut au lieu de `<xsl:element name="...">` qui crée un élément (comparer les lignes 30–56 dans les fichiers `isotab1-bis.xsl` et `isotab2.xsl`).

En utilisant `lotusxsl` avec le fichier source `frisotab-in.xml` nous obtenons la représentation alternative désirée.

```
java com.lotus.xsl.Process -in frisotab-in.xml -xsl isotab2.xsl -out frisotab2.xml
========= Parsing and preparing isotab2.xsl ==========
Parsing and init of isotab2.xsl took 1125 milliseconds
========= Parsing frisotab-in.xml ==========
Parse of frisotab-in.xml took 354 milliseconds
==============================
Transforming frisotab-in.xml via Input XSL...
transform took 18679 milliseconds
Total time took 19065 milliseconds
XSLProcessor: done
```

Voici le début du fichier `frisotab2.xml` :

```
1  <countries>
2  <country shortname="AD"
3           fullname="Andorre"
4           isocountry="la Principauté d'Andorre"
5           capital="Andorre-la-Vieille"
6           citizen="Andorran"
7           adjective="andorran"
8           currency="peseta; franc français"
9           isocurrency="ESP; FRF"
10          currensubunit="centimo; centime"/>
11 </countries>
```

Ce fichier contient la même information dans les attributs de l'élément `country` que le fichier `frisotab1.xml` dans les éléments contenus dans le même élément `country`. Remarquons que, pour une transparence totale, les attributs portent les mêmes noms que les types d'élément correspondants.

Avant de montrer comment utiliser cette petite base de données, nous voulons montrer une feuille de style qui permet de transformer une représentation vers l'autre. Ainsi, le fichier `isotab2to1.xsl` transforme la deuxième forme basée sur les attributs dans la première basée sur les types d'élément. Voici son contenu :

```
1  <?xml version='1.0' encoding="ISO-8859-1"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3
4  <xsl:output method="xml" encoding="ISO-8859-1"/>
5
6  <xsl:template match="/">
7  <xsl:element name="countries"><xsl:text>&#xA;</xsl:text>
8  <xsl:for-each select="countries/country">
9  <xsl:element name="country"><xsl:text>&#xA;</xsl:text>
```

```
<xsl:apply-templates select="{.}"/>
<xsl:element name="shortname">
<xsl:value-of select="@shortname"/>
</xsl:element>
<xsl:element name="fullname">
<xsl:value-of select="@fullname"/>
</xsl:element>
<xsl:element name="isocountry">
<xsl:value-of select="@isocountry"/>
</xsl:element>
<xsl:element name="capital">
<xsl:value-of select="@capital"/>
</xsl:element>
<xsl:element name="citizen">
<xsl:value-of select="@citizen"/>
</xsl:element>
<xsl:element name="adjective">
<xsl:value-of select="@adjective"/>
</xsl:element>
<xsl:element name="currency">
<xsl:value-of select="@currency"/>
</xsl:element>
<xsl:element name="isocurrency">
<xsl:value-of select="@isocurrency"/>
</xsl:element>
<xsl:element name="currensubunit">
<xsl:value-of select="@currensubunit"/>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:for-each>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Des retours en ligne (`<xsl:text>
</xsl:text>`, c'est-à-dire : le code littéral 10, `A` en hexadécimal) sont introduites ici et là pour améliorer la lisibilité du fichier. Nous avons vérifié que nous obtenons bien un fichier identique après traitement par un processeur XSL.

Inversément, nous transformons la première forme dans la deuxième avec le fichier `isotab1to2.xsl` qui suit.

```
<?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" encoding="ISO-8859-1"/>

<xsl:template match="/">
<xsl:element name="countries"><xsl:text>&#xA;</xsl:text>
<xsl:for-each select="countries/country">
<xsl:element name="country">
<xsl:attribute name="shortname">
<xsl:value-of select="shortname"/>
</xsl:attribute>
<xsl:attribute name="fullname">
<xsl:value-of select="fullname"/>
</xsl:attribute>
<xsl:attribute name="isocountry">
<xsl:value-of select="isocountry"/>
```

```
18  </xsl:attribute>
19  <xsl:attribute name="capital">
20  <xsl:value-of select="capital"/>
21  </xsl:attribute>
22  <xsl:attribute name="citizen">
23  <xsl:value-of select="citizen"/>
24  </xsl:attribute>
25  <xsl:attribute name="adjective">
26  <xsl:value-of select="adjective"/>
27  </xsl:attribute>
28  <xsl:attribute name="currency">
29  <xsl:value-of select="currency"/>
30  </xsl:attribute>
31  <xsl:attribute name="isocurrency">
32  <xsl:value-of select="isocurrency"/>
33  </xsl:attribute>
34  <xsl:attribute name="currensubunit">
35  <xsl:value-of select="currensubunit"/>
36  </xsl:attribute>
37  </xsl:element><xsl:text>&#xA;</xsl:text>
38  </xsl:for-each>
39  </xsl:element><xsl:text>&#xA;</xsl:text>
40  </xsl:template>
41
42  </xsl:stylesheet>
```

Notons le haut degré de symétrie entre ces deux fichiers où à l'intérieur de la boucle `for-each` (ligne 8–39 dans `isotab2to1.xsl` et 8–38 dans `isotab1to2.xsl`) où il suffit pratiquement d'interchanger les attributs et les éléments dans les constructeurs (`xsl:attribute` et `xsl:element`) et d'ajouter le signe « `@` » dans les références (`xsl:value-of`).

Évidemment, pour les francophones, nous pouvons traduire les noms des types d'élément. Voici une transformation `isotab1to1fr.xsl` de la première forme (utilisant des éléments) qui traduit les noms d'élément de l'anglais en français.

```
1   <?xml version='1.0' encoding="ISO-8859-1"?>
2   <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3
4   <xsl:output method="xml" encoding="ISO-8859-1"/>
5
6   <xsl:template match="/">
7   <xsl:element name="listeDesPays"><xsl:text>&#xA;</xsl:text>
8   <xsl:for-each select="countries/country">
9   <xsl:element name="pays"><xsl:text>&#xA;</xsl:text>
10  <xsl:element name="codeISO">
11  <xsl:value-of select="shortname"/>
12  </xsl:element><xsl:text>&#xA;</xsl:text>
13  <xsl:element name="nomComplet">
14  <xsl:value-of select="fullname"/>
15  </xsl:element><xsl:text>&#xA;</xsl:text>
16  <xsl:element name="nomISO">
17  <xsl:value-of select="isocountry"/>
18  </xsl:element><xsl:text>&#xA;</xsl:text>
19  <xsl:element name="capitale">
20  <xsl:value-of select="capital"/>
21  </xsl:element><xsl:text>&#xA;</xsl:text>
22  <xsl:element name="citoyen">
23  <xsl:value-of select="citizen"/>
```

```
</xsl:element><xsl:text>&#xA;</xsl:text>
<xsl:element name="adjectif">
<xsl:value-of select="adjective"/>
</xsl:element><xsl:text>&#xA;</xsl:text>
<xsl:element name="monnaie">
<xsl:value-of select="currency"/>
</xsl:element><xsl:text>&#xA;</xsl:text>
<xsl:element name="monnaieCodeISO">
<xsl:value-of select="isocurrency"/>
</xsl:element><xsl:text>&#xA;</xsl:text>
<xsl:element name="monnaieSousunité">
<xsl:value-of select="currensubunit"/>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:for-each>
</xsl:element><xsl:text>&#xA;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Nous avons introduit des retours à la ligne (en utilisant des appels numériques de caractères `
`) suivant chaque élément pour plus de lisibilité et nous avons utilisé la notation « chameau » pour le nom des types d'élément. La notation chameau (de l'anglais *camel*) est très répandue dans le domaine de la programmation orientée objets. Elle a pour but d'améliorer la lisibilité des codes-sources informatiques en signalant le début de chaque composante d'un mot-clef avec une majuscule.

Notre base de données prend maintenant la forme détaillée ci-dessous. Grâce à la notation chameau et les noms français, le fichier devient facile à lire.

```
<listeDesPays>
<pays>
<codeISO>AD</codeISO>
<nomComplet>Andorre</nomComplet>
<nomISO>la Principauté d'Andorre</nomISO>
<capitale>Andorre-la-Vieille</capitale>
<citoyen>Andorran</citoyen>
<adjectif>andorran</adjectif>
<monnaie>peseta; franc français</monnaie>
<monnaieCodeISO>ESP; FRF</monnaieCodeISO>
<monnaieSousInité>centimo; centime</monnaieSousunité>
</pays>
```

6.2.4. Utiliser la base de données

Il est temps maintenant de voir comment nous pouvons utiliser l'information qui se trouve dans notre base de données. Par exemple, si nous voulons obtenir une liste de tous les pays avec leur nom (simple et ISO) et leur capitale nous pouvons utiliser le fichier `frisotab1exa1.xsl` XSL suivant :

```
<?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="text" encoding="ISO-8859-1"/>
```

```
5
6  <xsl:template match="/listeDesPays">
7  <xsl:for-each select="pays">
8  <xsl:value-of select="nomComplet"/>
9  <xsl:text> (</xsl:text>
10 <xsl:value-of select="nomISO"/>
11 <xsl:text>) et la capitale </xsl:text>
12 <xsl:value-of select="capitale"/>
13 <xsl:text>.&#xA;</xsl:text><!-- retour à la ligne -->
14 </xsl:for-each>
15 </xsl:template>
16
17 </xsl:stylesheet>
```

Nous sélectionnons l'élément racine `listeDesPays` en ligne 6. Puis nous commençons une boucle de type `for-each`. Cette boucle va de la ligne 7 à la ligne 14. Pour chaque élément `pays`, on extrait son nom complet (ligne 8), suivi d'une parenthèse ouverte (ligne 9), suivi de son nom ISO (ligne 10), suivi de la chaîne de caractère « `) et la capitale` » (ligne 11), suivi du nom de la capitale (ligne 12) et on termine avec un retour à la ligne (ligne 13).

Le fichier texte résultant de la transformation commence ainsi :

```
1  Andorre (la Principauté d'Andorre) et la capitale Andorre-la-Vieille.
2  les Émirats arabes unis (les Émirats arabes unis) et la capitale Abou Dhabi.
3  l'Afghanistan (l'Émirat islamique d'Afghanistan) et la capitale Kaboul.
4  Antigua-et-Barbuda (Antigua-et-Barbuda) et la capitale Saint John's.
```

Nous pouvons permuter des entrées, les trier par ordre alphabétique et améliorer le traitement de l'article partitif « de, de la, du, des, l' ». C'est ce que nous réalisons dans le fichier XSL `frisotab1exa2.xsl` suivant :

```
1  <?xml version='1.0' encoding="ISO-8859-1"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3
4  <xsl:output method="text" encoding="ISO-8859-1"/>
5
6  <xsl:template match="/listeDesPays">
7  <xsl:for-each select="pays">
8   <xsl:sort select="capitale"/>
9   <xsl:if test="string(capitale) != '-'">
10   <xsl:value-of select="capitale"/>
11   <xsl:variable name="Nom" select="string(nomComplet)"/>
12   <xsl:choose>
13    <xsl:when test="substring($Nom,1,3) = 'les'">
14     <xsl:text> est la capitale des </xsl:text>
15     <xsl:value-of select="substring-after($Nom,' ')"/>
16    </xsl:when>
17    <xsl:when test="substring($Nom,1,3) = 'le '">
18     <xsl:text> est la capitale du </xsl:text>
19     <xsl:value-of select="substring-after($Nom,' ')"/>
20    </xsl:when>
21    <xsl:when test="substring($Nom,1,3) = 'la '">
22     <xsl:text> est la capitale de </xsl:text>
23     <xsl:value-of select="$Nom"/>
24    </xsl:when>
```

```
25      <xsl:when test='substring($Nom,1,2) = "l&#39;"'>
26       <xsl:text> est la capitale de </xsl:text>
27       <xsl:value-of select="$Nom"/>
28      </xsl:when>
29      <xsl:otherwise>
30       <xsl:variable name="L1" select="substring($Nom,1,1)"/>
31       <xsl:choose>
32        <xsl:when test="$L1 = 'A' or $L1 = 'E' or $L1 = 'I' or
33                        $L1 = 'O' or $L1 = 'U' or $L1 = 'Y'">
34         <xsl:text> est la capitale de l'</xsl:text>
35        </xsl:when>
36        <xsl:otherwise>
37         <xsl:text> est la capitale de </xsl:text>
38        </xsl:otherwise>
39       </xsl:choose>
40       <xsl:value-of select="$Nom"/>
41      </xsl:otherwise>
42     </xsl:choose>
43     <xsl:text>.&#xA;</xsl:text><!-- retour à la ligne -->
44    </xsl:if>
45  </xsl:for-each>
46  </xsl:template>
47
48  </xsl:stylesheet>
```

Comme avec la feuille de style précédente `frisotab1exa2.xsl` nous avons une boucle `xsl:foreach` sur tous les éléments de type `pays` (lignes 7–45). En ligne 8 nous déclarons vouloir que la sortie soit triée par ordre alphabétique en utilisant le contenu des éléments `capitale` comme clé de tri. Avant d'aller plus loin, nous rejetons tous les pays qui n'ont pas une valeur adéquate pour leur capitale (test `xsl:if` de la ligne 9 qui vérifie que la valeur est bien différente d'un tiret `'-'`). Une variable `Nom` est déclarée en ligne 11 et mise égale au contenu de l'élément `nomComplet` interprété comme une chaîne de caractères.

Nous devons traiter ensuite les différents cas de figure pour la forme de l'article partitif (instruction `xsl:choose` qui couvre les lignes 12–42). Les instructions `xsl:when` testent la valeur des premiers caractères de la variable `Nom` qui contient le début de la chaîne de caractères représentant le nom complet du pays. En fonction de ce que nous trouvons, nous générons la forme grammaticalement correcte. Un cas intéressant est la forme « l' » où nous devons faire attention à bien imbriquer les différentes formes des caractères « ' » et « " » dans le test pour l'attribut `select` (voir la ligne 25 qui utilise la représentation décimale de l'apostrophe « ' »). Ensuite, si nous n'avons pas trouvé d'article, nous nous retrouvons dans la branche `xsl:otherwise` (lignes 29–41). Ici, nous devons à nouveau distinguer deux cas : le nom commence par une voyelle (test des lignes 32–33) ou pas (branche`xsl:otherwise` des lignes 36–38). Cet exemple montre comment XSL permet la génération d'un format de sortie à la demande, en particulier grâce à la possibilité de construire des tests assez complexes.

Le résultat de l'application de cette feuille de style à notre base de données liste les capitales en ordre alphabétique. Le début du fichier résultant suit :

```
1  Abou Dhabi est la capitale des Émirats arabes unis.
2  Abuja est la capitale du Nigeria.
```

```
Accra est la capitale du Ghana.
Achgabat  est la capitale du Turkménistan.
Adamstown est la capitale des Iles Pitcairn.
Addis-Abeba est la capitale de l'Éthiopie.
Alger est la capitale de l'Algérie.
Alofi est la capitale de Nioué.
```

Nous pouvons aussi utiliser notre base de données pour générer du HTML (ou du LaTeX). Voici comment générer une table HTML qui montre une liste des monnaies triée alphabétiquement, avec la sous-unité, l'abréviation ISO et le pays où cette monnaie a cours. Pour plus de variété, nous utilisons la deuxième forme de la base de données où l'information est codée dans les attributs. La transformation entre les deux formats s'est faite à l'aide du fichier XSL `frisotab1to2.xsl` suivant :

```
<?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" encoding="ISO-8859-1"/>

<xsl:template match="listeDesPays">
<xsl:copy>
<xsl:apply-templates/>
</xsl:copy><xsl:text>&#xa;</xsl:text>
</xsl:template>

<xsl:template match="pays">
<xsl:copy>
<xsl:for-each select="*">
<xsl:attribute name="{name(.)}"><xsl:value-of select="."/>
</xsl:attribute>
</xsl:for-each>
</xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Les lignes 6–10 choisissent l'élément racine `listeDesPays` et copient à l'aide de l'instruction `xsl:copy` tout son contenu. Puis pour chaque élément `pays` (lignes 12–19) nous copions son contenu et nous faisons une boucle sur tous les éléments enfant « `*` » (lignes 14–17) en générant, ligne 15, un attribut avec le même nom que l'élément en question (`name="{name(.)}"`) avec comme valeur le contenu de ce même élément (`<xsl:value-of select="."/>`). Ainsi ce fichier est un équivalent plus court du fichier `isotab1to2.xsl` de la section 6.2.3.

Le fichier `frisotab2exa3.xsl` qui construit la table HTML est le suivant :

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
<body bgcolor="white">
<h1>Les monnaies et les pays</h1>
<table border="1" rules="rows">
<thead>
<tr>
```

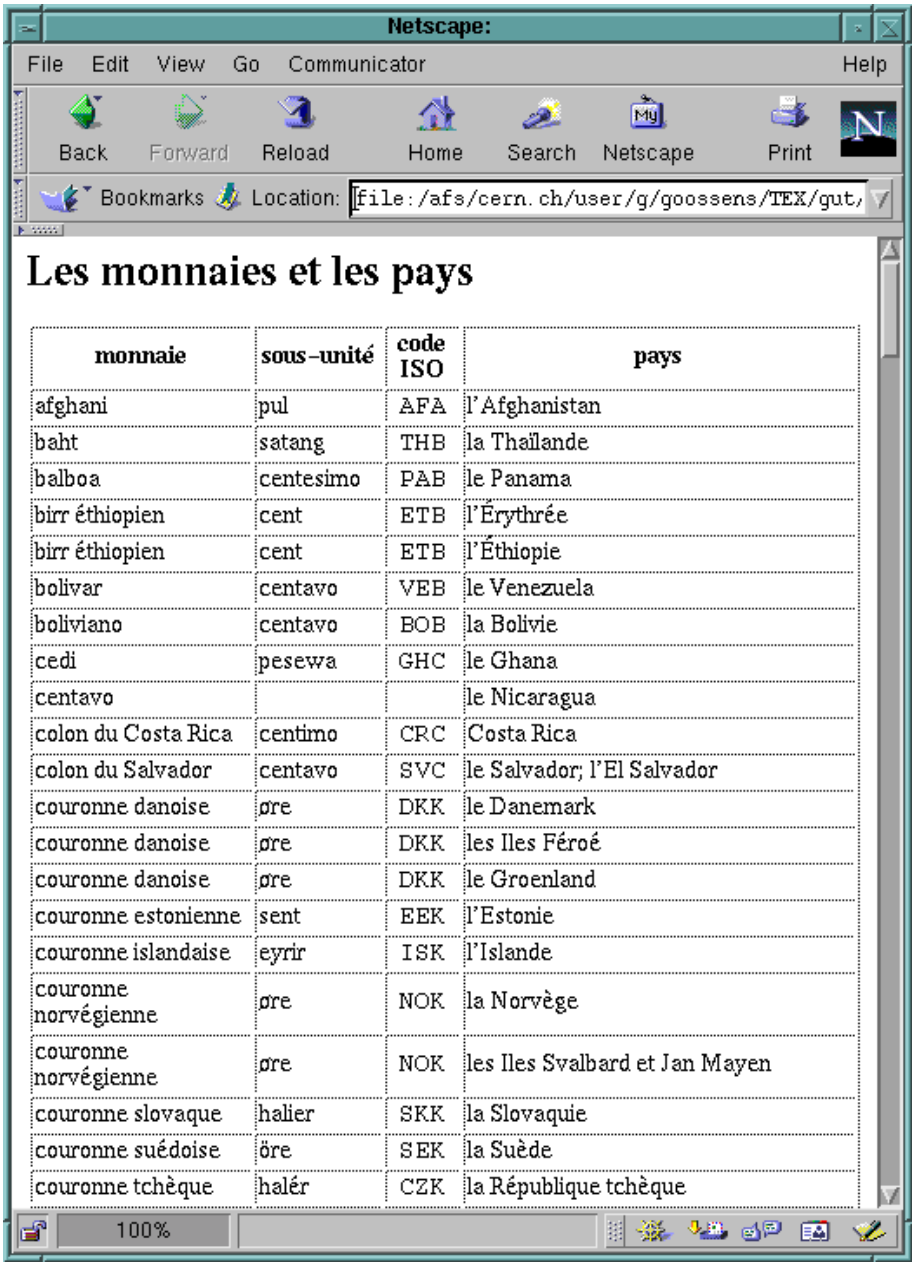
```
8  <td align="center"><b>monnaie</b></td>
9  <td align="center"><b>sous-unité</b></td>
10 <td align="center"><b>code ISO</b></td>
11 <td align="center"><b>pays</b></td>
12 </tr>
13 </thead>
14 <xsl:for-each select="/listeDesPays/pays">
15 <xsl:sort select="@monnaie"/>
16 <xsl:if test="string(@monnaie) != '-'">
17 <tr>
18 <td><xsl:value-of select="@monnaie"/></td>
19 <td><xsl:value-of select="@monnaieSousunité"/></td>
20 <td align="center"><tt><xsl:value-of select="@monnaieCodeISO"/></tt></td>
21 <td><xsl:value-of select="@nomComplet"/></td>
22 </tr>
23 </xsl:if>
24 </xsl:for-each>
25 </table>
26 </body>
27 </html>
```

Comme XML est avant tout un langage pour l'internet la spécification XSLT (à sa section 2.3.1) propose une approche simplifiée pour générer un fichier HTML dans le cas où l'arbre résultant peut être généré à l'aide d'un seule modèle qui agit sur l'élément racine. C'est ce que nous avons mis en œuvre dans notre feuille de style, où, ligne 2, nous déclarons que le domaine nominal par défaut est `html` ce qui nous permet d'insérer des instructions HTML sans préfixe de domaine nominal. Nous spécifions le titre (ligne 4) et construisons la première partie de la table (lignes 5–13). Puis commence une boucle de type `xsl:foreach` sur les élément `/listeDesPays/pays` (lignes 14–24). Le résultat de la boucle est trié par ordre alphabétique du contenu de l'attribut `monnaie` (ligne 15), mais en éliminant les cas où cette valeur serait non-valable (test en ligne 16). Finalement nous composons une ligne de la table (lignes 17–22) en remplissant la première colonne avec le nom de la monnaie (ligne 18), la deuxième avec le nom de sa sous-unité (ligne 19), la troisième avec le code ISO centré et en police chasse fixe (ligne 20) et la quatrième colonne avec le nom complet du pays (ligne 21). La figure 10 montre le résultat HTML.

6.3. Un recueil de poèmes

Dans ce qui précède nous avons traité des fichiers relativement simples. Dans cette section, nous étudions comment combiner plusieurs fichiers source à l'entrée et comment générer plusieurs fichiers à la sortie. Comme fichiers source, nous avons choisi quelques poèmes de Paul Verlaine que nous avons trouvés sur un site français à l'URL `http://www.ambafrance.org/FLORILEGE/verlaine/a.html`.

La communauté XML (ou SGML) a développé plusieurs DTD « standard » pour être utilisées dans leurs domaines respectifs. Mentionnons d'abord Docbook[36, 9], qui est la base du balisage de beaucoup de manuels d'informatique. Pour notre exemple nous nous tournons plutôt vers les sciences humaines, où la famille des DTD du projet « TEI » (*Text Encoding Initiative*[4, 33]) est très répandue.



Les monnaies et les pays

monnaie	sous-unité	code ISO	pays
afghani	pul	AFA	l'Afghanistan
baht	satang	THB	la Thaïlande
balboa	centesimo	PAB	le Panama
birr éthiopien	cent	ETB	l'Érythrée
birr éthiopien	cent	ETB	l'Éthiopie
bolivar	centavo	VEB	le Venezuela
boliviano	centavo	BOB	la Bolivie
cedi	pesewa	GHC	le Ghana
centavo			le Nicaragua
colon du Costa Rica	centimo	CRC	Costa Rica
colon du Salvador	centavo	SVC	le Salvador; l'El Salvador
couronne danoise	øre	DKK	le Danemark
couronne danoise	øre	DKK	les Iles Féroé
couronne danoise	øre	DKK	le Groenland
couronne estonienne	sent	EEK	l'Estonie
couronne islandaise	eyrir	ISK	l'Islande
couronne norvégienne	øre	NOK	la Norvège
couronne norvégienne	øre	NOK	les Iles Svalbard et Jan Mayen
couronne slovaque	halier	SKK	la Slovaquie
couronne suédoise	öre	SEK	la Suède
couronne tchèque	halér	CZK	la République tchèque

FIGURE 10 – *Une table montrant monnaies et pays*

Nous avons balisé quelques poèmes extraits du recueil *Poèmes saturniens* dans le fichier `verlaineps.xml` :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<div1>
<head>Poèmes saturniens (1866)</head>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<div2>
<head><hi rend="ital">Soleils couchants</hi></head>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<lg type="stanza" org="uniform" sample="complete" part="N">
<l>Une aube affaiblie</l>
<l>Verse par les champs</l>
<l>La mélancolie</l>
<l>Des soleils couchants.</l>
<l>La mélancolie</l>
<l>Berce de doux chants</l>
<l>Mon c&oelig;ur qui s'oublie</l>
<l>Aux soleils couchants.</l>
<l>Et d'étranges rêves,</l>
<l>Comme des soleils</l>
<l>Couchants sur les grèves,</l>
<l>Fantômes vermeils,</l>
<l>Défilent sans trêves,</l>
<l>Défilent, pareils</l>
<l>À de grands soleils</l>
<l>Couchants sur les grèves.</l>
</lg>
</div2>
<!--++++++++++++++++++++++++++++++++ -->
<div2>
<head><hi rend="ital">Mon rêve familier</hi></head>
<lg type="stanza" org="uniform" sample="complete" part="N">
<l>Je fais souvent ce rêve étrange et pénétrant</l>
<l>D'une femme inconnue, et que j'aime, et qui m'aime,</l>
<l>Et qui n'est, chaque fois, ni tout à fait la même</l>
<l>Ni tout à fait une autre, et m'aime et me comprend.</l>
</lg>
<lg>
<l>Car elle me comprend, et mon c&oelig;ur, transparent</l>
<l>Pour elle seule, hélas ! cesse d'être un problème</l>
<l>Pour elle seule, et les moiteurs de mon front blême,</l>
<l>Elle seule les sait rafraîchir, en pleurant.</l>
</lg>
<lg>
<l>Est-elle brune, blonde ou rousse ? &dash; Je l'ignore.</l>
<l>Son nom ? Je me souviens qu'il est doux et sonore</l>
<l>Comme ceux des aimés que la Vie exila.</l>
</lg>
<lg>
<l>Son regard est pareil au regard des statues,</l>
<l>Et, pour sa voix, lointaine, et calme, et grave, elle a</l>
<l>L'inflexion des voix chères qui se sont tues.</l>
</lg>
</div2>
<!-- ++++++++++++++++++++++++++++++++++++ -->
<div2>
<head><hi rend="ital">Nevermore</hi></head>
<lg type="stanza" org="uniform" sample="complete" part="N">
<l>Souvenir, souvenir, que me veux-tu ? L'automne</l>
<l>Faisait voler la grive à travers l'air atone,</l>
<l>Et le soleil dardait un rayon monotone</l>
```

```
<l>Sur le bois jaunissant où la bise détone.</l>
</lg>
<lg>
<l>Nous étions seul à seule et marchions en rêvant,</l>
<l>Elle et moi, les cheveux et la pensée au vent.</l>
<l>Soudain, tournant vers moi son regard émouvant :</l>
<l>ń Quel fut ton plus beau jour ? ż fit sa voix d'or vivant,</l>
</lg>
<lg>
<l>Sa voix douce et sonore, au frais timbre angélique.</l>
<l>Un sourire discret lui donna la réplique,</l>
<l>Et je baisai sa main blanche, dévotement.</l>
</lg>
<lg>
<l>&dash; Ah ! les premières fleurs, qu'elles sont parfumées !</l>
<l>Et qu'il bruit avec un murmure charmant</l>
<l>Le premier ń oui ż qui sort de lèvres bien-aimées !</l>
</lg>
</div2>
<!-- ++++++++++++++++++++++++++++++++++++ -->
<div2>
<head><hi rend="ital">Chanson d'automne</hi></head>
<lg type="stanza" org="uniform" sample="complete" part="N">
<l>Les sanglots longs</l>
<l>Des violons</l>
<l rend="indent">De l'automne</l>
<l>Blessent mon c&oelig;ur</l>
<l>D'une langueur</l>
<l rend="indent">Monotone.</l>
</lg>
<lg>
<l>Tout suffocant</l>
<l>Et blême, quand</l>
<l rend="indent">Sonne l'heure,</l>
</lg>
<lg>
<l>Je me souviens</l>
<l>Des jours anciens</l>
<l rend="indent">Et je pleure ;</l>
</lg>
<lg>
<l>Et je m'en vais</l>
<l>Au vent mauvais</l>
<l rend="indent">Qui m'emporte</l>
<l>Deçà, delà,</l>
<l>Pareil à la</l>
<l rend="indent">Feuille morte.</l>
</lg>
</div2>
</div1>
```

Le recueil se trouve dans un élément de type `div1` (lignes 2–109) et chaque poème est à l'intérieur d'un élément de type `div2` (lignes 2–26, 28–52, etc.). Le titre d'un poème est saisi dans un élément de type `head`, où nous voyons également l'élément `hi` pour spécifier le type de mise en évidence (lignes 6, 29, etc.). Les stances sont délimitées par des éléments `lg` (ligne 8–25, 30–35, 36–41, . . .) et les lignes individuelles par les éléments `l`. Parfois, nous indiquons que nous désirons un renfoncement pour une ligne donnée à l'aide de l'attribut `rend` (lignes 85, 88, etc.).

Avec ces mêmes conventions, nous avons également saisi quelques poèmes du recueil *Fêtes galantes* dans le fichier `verlainefg.xml` :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<div1>
<head>Fêtes galantes (1869)</head>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<div2>
<head><hi rend="ital">Clair de lune</hi></head>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<lg type="stanza" org="uniform" sample="complete" part="N">
<l>Votre âme est un paysage choisi</l>
<l>Que vont charmant masques et bergamasques,</l>
<l>Jouant du luth, et dansant, et quasi</l>
<l>Tristes sous leurs déguisements fantasques. </l>
</lg><lg>
<l>Tout en chantant sur le mode mineur</l>
<l>L'amour vainqueur et la vie opportune,</l>
<l>Ils n'ont pas l'air de croire à leur bonheur</l>
<l>Et leur chanson se mêle au clair de lune,</l>
</lg><lg>
<l>Au calme clair de lune triste et beau,</l>
<l>Qui fait rêver les oiseaux dans les arbres </l>
<l>Et sangloter d'extase les jets d'eau,</l>
<l>Les grands jets d'eau sveltes parmi les marbres.</l>
</lg>
</div2>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<div2>
<head><hi rend="ital">Les indolents</hi></head>
<lg type="stanza" org="uniform" sample="complete" part="N">
<l>Bah ! malgré les destins jaloux,</l>
<l>Mourons ensemble, voulez-vous ?</l>
<l>&dash; La proposition est rare.</l>
</lg><lg>
<l>&dash; Le rare est bon. Donc mourons</l>
<l>Comme dans les Décamérons.</l>
<l>&dash; Hi ! Hi ! Hi ! quel amant bizarre !</l>
</lg><lg>
<l>&dash; Bizarre, je ne sais. Amant</l>
<l>Irréprochable, assurément.</l>
<l>Si vous voulez, mourons ensemble ?</l>
</lg><lg>
<l>&dash; Monsieur, vous raillez mieux encor</l>
<l>Que vous n'aimez, et parlez d'or ;</l>
<l>Mais taisons-nous, si bon vous semble ! ż </l>
</lg><lg>
<l>Si bien que ce soir-là Tircis</l>
<l>Et Dorimène, à deux assis</l>
<l>Non loin de deux sylvains hilares,</l>
</lg><lg>
<l>Eurent l'inexpiable tort</l>
<l>D'ajourner une exquise mort.</l>
<l>Hi ! hi ! hi ! les amants bizarres !</l>
</lg>
</div2>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<div2>
<head><hi rend="ital">Colloque sentimental</hi></head>
<lg type="stanza" org="uniform" sample="complete" part="N">
<l>Dans le vieux parc solitaire et glacé,</l>
<l>Deux formes ont tout à l'heure passé.</l>
</lg><lg>
```

```
<l>Leurs yeux sont morts et leurs lèvres sont molles,</l>
<l>Et l'on entend à peine leurs paroles.</l>
</lg><lg>
<l>Dans le vieux parc solitaire et glacé,</l>
<l>Deux spectres ont évoqué le passé.</l>
</lg><lg>
<l>&dash; Te souvient-il de notre extase ancienne ?</l>
<l>&dash; Pourquoi voulez-vous donc qu'il m'en souvienne ?</l>
</lg><lg>
<l>&dash; Ton c&oelig;ur bat-il toujours à mon seul nom ?</l>
<l>Toujours vois-tu mon âme en rêve ? &dash; Non.</l>
</lg><lg>
<l>&dash; Ah ! les beaux jours de bonheur indicible</l>
<l>Où nous joignons nos bouches ! &dash; C'est possible.</l>
</lg><lg>
<l>&dash; Qu'il était bleu, le ciel, et grand, l'espoir !</l>
<l>&dash; L'espoir a fui, vaincu, vers le ciel noir.</l>
</lg><lg>
<l>Tels ils marchaient dans les avoines folles,</l>
<l>Et la nuit seule entendit leurs paroles.</l>
</lg>
</div2>
</div1>
```

Nous combinons les deux recueils à l'aide du fichier principal `verlainefg.xml` :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE TEI.2 SYSTEM "teixlite.dtd" [
<!ENTITY dash "&#x2010;">
<!ENTITY mdash "&#x2014;">
<!ENTITY oelig "&#x0153;">
<!ENTITY Poèmes-saturniens SYSTEM "verlaineps.xml">
<!ENTITY Fêtes-galantes    SYSTEM "verlainefg.xml">
]>
<TEI.2>
<teiHeader type="text" status="new">
<fileDesc>
<titleStmt>
<title>Divers recueils de poèmes</title>
<author>Verlaine, Paul</author>
<respStmt>
<resp>Transcription en code TEI.2</resp>
<name>Michel Goossens</name></respStmt>
</titleStmt>
<publicationStmt>
<distributor>Distributed by mg for GUTenberg</distributor>
</publicationStmt>
<sourceDesc default="NO">
<p>http://www.ambafrance.org/FLORILEGE/verlaine/a.html</p>
</sourceDesc>
</fileDesc>
<profileDesc>
<langUsage default="NO">
<language id="FR">Français</language>
</langUsage>
</profileDesc>
</teiHeader>
  <text>
<front>
<titlePage>
<docTitle>
<titlePart>Recueil de poèmes de Paul Verlaine</titlePart>
```

```
</docTitle>
<docAuthor>Michel Goossens</docAuthor>
<docDate>Octobre 1999</docDate>
</titlePage>
</front>
<body>
&Poèmes-saturniens;
&Fêtes-galantes;
</body>
</text>
</TEI.2>
```

Ce fichier définit (lignes 6 et 7) des entités externes pour insérer les deux fichiers sources `verlaineps.xml` (ligne 43) et `verlainefg.xml` (ligne 44). Nous faisons référence à la DTD XML simplifiée pour la TEI (`teixlite.dtd` en ligne 2). Dans le sous-ensemble interne de la DTD, nous définissons quelques autres entités : les tirets `ndash` et `mdash` et, plus important pour le français, la ligature œ, qui, absente du codage ISO-8859-1, ne peut donc être saisie directement. Tout ce qui se trouve à l'intérieur de l'en-tête TEI correspondant à l'élément `teiHeader` (lignes 10–31) sera ignoré dans notre traitement ultérieur. Ce qui nous intéresse se trouvent dans l'élément `text` (lignes 32–46).

6.3.1. TEX comme formateur avec PassiveTEX

Sebastian RAHTZ a développé une série de feuilles de style XSL pour transformer les fichiers XML codés d'après la DTD TEI-lite en TEX et HTML[31].

```
java com.jclark.xsl.sax.Driver verlainetei.xml tei.xsl verlainetei.fo
```

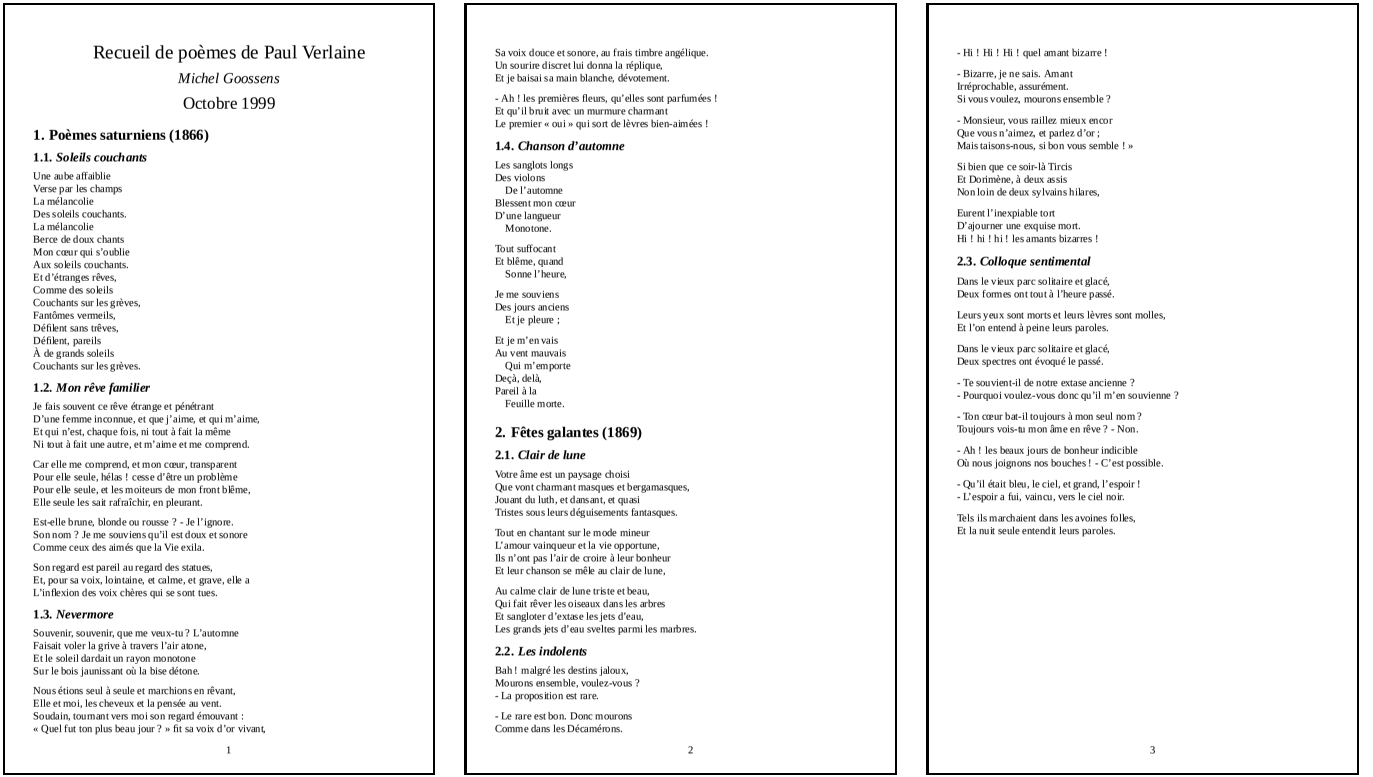
En particulier, nous pouvons traiter le fichier `verlaine.fo` contenant des objets de formatage XSLF avec PassiveTEX[29], une adaptation de TEX également par Sebastian RAHTZ qui génère du PDF. Ainso nous générons un fichier `verlainetei.pdf` (voir figure 11).

6.3.2. Création d' une toile de fichiers HTML

Nous nous proposons, à partir des mêmes fichiers XML contentant des poèmes de Verlaine, de créer une toile HTML (un ensemble de fichiers, un par poème, complété d'un fichier principal qui permet d'adresser tous les fichiers individuels à l'aide des liens hypertexte).

Voici le fichier de style XSL `htmlsplit.xsl` qui gère cette transformation.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY nbsp   " ">
]>
<xsl:stylesheet version="1.0"
```



Recueil de poèmes de Paul Verlaine

Michel Goossens

Octobre 1999

1. Poèmes saturniens (1866)

1.1. *Soleils couchants*

Une aube affaiblie
Verse par les champs
La mélancolie
Des soleils couchants.
La mélancolie
Berce de doux chants
Mon cœur qui s'oublie
Aux soleils couchants.
Et d'étranges rêves,
Comme des soleils
Couchants sur les grèves,
Fantômes vermeils,
Défilent sans trêves,
Défilent, pareils
À de grands soleils
Couchants sur les grèves.

1.2. *Mon rêve familier*

Je fais souvent ce rêve étrange et pénétrant
D'une femme inconnue, et que j'aime, et qui m'aime,
Et qui n'est, chaque fois, ni tout à fait la même
Ni tout à fait une autre, et m'aime et me comprend.

Car elle me comprend, et mon cœur, transparent
Pour elle seule, hélas ! cesse d'être un problème
Pour elle seule, et les moiteurs de mon front blême,
Elle seule les sait rafraîchir, en pleurant.

Est-elle brune, blonde ou rousse ? - Je l'ignore.
Son nom ? Je me souviens qu'il est doux et sonore
Comme ceux des aimés que la Vie exila.

Son regard est pareil au regard des statues,
Et, pour sa voix, lointaine, et calme, et grave, elle a
L'inflexion des voix chères qui se sont tues.

1.3. *Nevermore*

Souvenir, souvenir, que me veux-tu ? L'automne
Faisait voler la grive à travers l'air atone,
Et le soleil dardait un rayon monotone
Sur le bois jaunissant où la bise détone.

Nous étions seul à seule et marchions en rêvant,
Elle et moi, les cheveux et la pensée au vent.
Soudain, tournant vers moi son regard émouvant :
« Quel fut ton plus beau jour ? » fit sa voix d'or vivant,

Sa voix douce et sonore, au frais timbre angélique.
Un sourire discret lui donna la réplique,
Et je baisai sa main blanche, dévotement.

- Ah ! les premières fleurs, qu'elles sont parfumées !
Et qu'il bruit avec un murmure charmant
Le premier « oui » qui sort de lèvres bien-aimées !

1.4. *Chanson d'automne*

Les sanglots longs
Des violons
  De l'automne
Blessent mon cœur
D'une langueur
  Monotone.

Tout suffocant
Et blême, quand
  Sonne l'heure,

Je me souviens
Des jours anciens
  Et je pleure ;

Et je m'en vais
Au vent mauvais
  Qui m'emporte
Deçà, delà,
Pareil à la
  Feuille morte.

2. Fêtes galantes (1869)

2.1. *Clair de lune*

Votre âme est un paysage choisi
Que vont charmant masques et bergamasques,
Jouant du luth, et dansant, et quasi
Tristes sous leurs déguisements fantasques.

Tout en chantant sur le mode mineur
L'amour vainqueur et la vie opportune,
Ils n'ont pas l'air de croire à leur bonheur
Et leur chanson se mêle au clair de lune,

Au calme clair de lune triste et beau,
Qui fait rêver les oiseaux dans les arbres
Et sangloter d'extase les jets d'eau,
Les grands jets d'eau sveltes parmi les marbres.

2.2. *Les indolents*

Bah ! malgré les destins jaloux,
Mourons ensemble, voulez-vous ?
- La proposition est rare.

- Le rare est bon. Donc mourons
Comme dans les Décamérons.

- Hi ! Hi ! Hi ! quel amant bizarre !

- Bizarre, je ne sais. Amant
Irréprochable, assurément.
Si vous voulez, mourons ensemble ?

- Monsieur, vous railler mieux encor
Que vous n'aimez, et parlez d'or ;
Mais taisons-nous, si bon vous semble ! »

Si bien que ce soir-là Tircis
Et Dorimène, à deux assis
Non loin de deux sylvains hilares,

Eurent l'inexpiable tort
D'ajourner une exquise mort.
Hi ! hi ! hi ! les amants bizarres !

2.3. *Colloque sentimental*

Dans le vieux parc solitaire et glacé,
Deux formes ont tout à l'heure passé.

Leurs yeux sont morts et leurs lèvres sont molles,
Et l'on entend à peine leurs paroles.

Dans le vieux parc solitaire et glacé,
Deux spectres ont évoqué le passé.

- Te souvient-il de notre extase ancienne ?
- Pourquoi voulez-vous donc qu'il m'en souvienne ?

- Ton cœur bat-il toujours à mon seul nom ?
Toujours vois-tu mon âme en rêve ? - Non.

- Ah ! les beaux jours de bonheur indicible
Où nous joignons nos bouches ! - C'est possible.

- Qu'il était bleu, le ciel, et grand, l'espoir !
- L'espoir a fui, vaincu, vers le ciel noir.

Tels ils marchaient dans les avoines folles,
Et la nuit seule entendit leurs paroles.

FIGURE 11 – *Extrait des* Poèmes saturniens *et des* Fêtes galantes *de Paul Verlaine.*
Cette sortie PDF est générée à partir d'un fichier balisé en XML utilisant la DTD TEI, transformé en objets de formatage par des feuilles de style XSL. Ces objets sont finalement traduits avec PassiveTEX en PDF avec la commande `pdftex "&fotex" verlainetei.fo`.

```
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:date="http://www.jclark.com/xt/java/java.util.Date"
      xmlns:xt="http://www.jclark.com/xt"
      extension-element-prefixes="xt">

<xsl:output method="html" encoding="UTF-8"/>

<xsl:variable name="MasterFile">verlaine</xsl:variable>

<!--
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  + Début des définitions des modèles nommées                          +
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-->

<xsl:template name="footer">
 <hr /><address>
  Dernière modification : Michel Goossens,
<xsl:choose>
  <xsl:when test="function-available('date:to-string') and
                  function-available('date:new')">
  <!-- format date : Fri Dec 31 23:59:59 PDT 1999 -->
  <!--               123456789012345678901234567 -->
  <xsl:variable name="datetemp" select="date:to-string(date:new())"/>
  <xsl:variable name="mois" select="substring($datetemp,5,3)"/>
  <xsl:variable name="jour" select="substring($datetemp,9,2)"/>
  <xsl:variable name="année" select="substring($datetemp,string-length($datetemp)-3,4)"/>
  <xsl:variable name="Date" select="concat($jour,' ',$mois,'. ',$année)"/>
  <xsl:value-of select="$Date"/>
  </xsl:when>
  <xsl:otherwise>
  30 Oct 1999
  </xsl:otherwise>
</xsl:choose>
 <br />
 </address>
</xsl:template>

<xsl:template name="header">
  <xsl:number format="1.1" level="multiple" count="div1|div2"/>.
  <xsl:value-of select="head"/>
</xsl:template>

<!-- Construction de pointeurs vers le poème suivant and précédent -->
<xsl:template name="xrefpanel">
  <xsl:param name="home"/>
<table align="center" rules="none">
<tr>
<td><a href="{$home}">Monter</a></td>
<td><xsl:choose>
  <xsl:when test="preceding-sibling::div2">
  <a href="{concat($MasterFile,generate-id(preceding-sibling::div2[1]))}.html">Précédent</a>
  </xsl:when>
  <xsl:when test="ancestor::div1/preceding-sibling::div1[1]/div2[last()]">
  <a href="{concat($MasterFile,generate-id(
        ancestor::div1/preceding-sibling::div1[1]/div2[last()]))}.html">Précédent</a>
  </xsl:when>
</xsl:choose></td>
<td><xsl:choose>
  <xsl:when test="following-sibling::div2">
  <a href="{concat($MasterFile,generate-id(following-sibling::div2[1]))}.html">Suivant</a>
  </xsl:when>
  <xsl:when test="ancestor::div1/following-sibling::div1[1]/div2[1]">
```

```
69    <a href="{concat($MasterFile,generate-id(
70          ancestor::div1/following-sibling::div1[1]/div2[1]))}.html">Suivant</a>
71    </xsl:when>
72  </xsl:choose></td>
73  </tr>
74  </table>
75  <h1><xsl:value-of select="ancestor::div1/head"/></h1>
76  <xsl:variable name="n" select="concat($MasterFile,generate-id())"/>
77  <h2><a name="{$n}"></a><xsl:value-of select="head"/></h2>
78  </xsl:template>
79
80  <!--
81    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
82    + Début des motifs de séléction pour les éléments                 +
83    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
84  -->
85
86  <xsl:template match="/">
87   <html>
88   <head>
89   <title><xsl:value-of select="//titlePart"/></title>
90   <link rel="stylesheet" type="text/css" href="verlaine.css"/>
91   </head>
92   <body>
93   <h1><xsl:value-of select="//titlePart"/></h1>
94   <h2><xsl:value-of select="//docAuthor"/></h2>
95   <h2><xsl:value-of select="//docDate"/></h2>
96   <xsl:for-each select="//div1">
97    <h3><xsl:call-template name="header"/></h3>
98    <ul>
99    <xsl:for-each select="div2">
100    <xsl:variable name="poemname" select="concat($MasterFile,concat(generate-id()),'.html')"/>
101    <li><a href="{$poemname}"><xsl:call-template name="header"/></a></li>
102   </xsl:for-each>
103  </ul>
104  </xsl:for-each>
105  <xsl:call-template name="footer"/>
106  </body>
107  </html>
108  <xsl:apply-templates select="//div2"/>
109 </xsl:template>
110
111 <xsl:template match="div2">
112   <xsl:variable name="file" select="concat($MasterFile,concat(generate-id()),'.html')"/>
113   <xt:document method="html" href="{$file}"  encoding="UTF-8">
114     <html>
115       <head>
116         <title><xsl:value-of select="head"/></title>
117         <link rel="stylesheet" type="text/css" href="verlaine.css"/>
118       </head>
119       <body>
120         <xsl:call-template name="xrefpanel">
121           <xsl:with-param name="home" select="concat($MasterFile,'.html')"/>
122         </xsl:call-template>
123         <xsl:apply-templates/>
124         <xsl:call-template name="footer"/>
125       </body>
126     </html>
127   </xt:document>
128 </xsl:template>
129
130 <xsl:template match="head">
131 <!-- ignorer le contenu -->
```

```
132 </xsl:template>
133
134 <xsl:template match="hi">
135   <xsl:apply-templates/>
136 </xsl:template>
137
138 <xsl:template match="lg">
139   <p><xsl:apply-templates/></p>
140 </xsl:template>
141
142 <xsl:template match="l">
143 <xsl:if test="@rend='indent'">
144  &amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;</xsl:if>
145   <xsl:apply-templates/>
146 <xsl:if test="not(position()=last())">
147 <br /></xsl:if>
148 </xsl:template>
149
150 <xsl:template match="*">
151 <p> ***** Début de l'élément non-prévu </p>
152 <xsl:apply-templates/>
153 <p> Fin de l'élément non-prévu ****** </p>
154 </xsl:template>
155
156 </xsl:stylesheet>
```

Cette feuille de style XSL est assez complexe et nous allons en examiner les points les plus importants. D'abord la ligne 7 introduit le domaine nominal `date` et la ligne 8 le domaine nominal `xt`, qui, de plus, est déclaré comme étant un préfixe d'élément d'extension. Ceci signifie que le code préfixé est une extension du processeur que nous utilisons (`xt` le cas échéant) et qu'il n'y a aucune garantie que notre feuille de style marche avec un autre processeur (comme `lotusxsl`), sauf si ce dernier inclut un support pour les mêmes fonctions ou éléments. Les extensions pour les processeurs se trouvent dans leurs documentations respectives (les sections *Extension Functions* et *Multiple Output Documents* dans [5] dans notre cas).

Nous utilisons la fonction d'extension pour générer la date (ligne 29), tout en nous assurant avant que les fonctions de date sont bien disponibles (lignes 25–26). Si elles ne le sont pas nous écrivons la chaîne de caractères « `30 Oct 1999` » (ligne 38). Les lignes 29–34 montrent quelques-unes des fonctions pour manipuler les chaînes de caractères (voir la section E pour une liste et un renvoi à la section de la spécification XPath où elles sont décrites). Ces même lignes montrent également comment utiliser les variables XSL.

Nous avons modularisé notre feuille de style XSL en définissant des modèles nommés (`footer` en lignes 21–42, `header` en lignes 44–47 et `xrefpanel` en lignes 50-76).

En ligne 11, nous déclarons que nous désirons écrire un fichier HTML ; le domaine nominal par défaut sera donc HTML. Ceci nous permet d'introduire les balises HTML sans utiliser de préfixe. Pour traiter correctement les ligatures `mdash` (—), `ndash` (–) et `oelig` (œ) nous choisissons le codage UTF-8.

Le traitement effectif commence avec le modèle pour la racine à la ligne 86. Nous construisons un titre global en utilisant la balise `TitlePart` dans le document source (ligne 89) et introduisons un lien vers un fichier de style CSS `verlaine.css` (ligne 90).

Nous générons le titre pour le corps du document et nous y ajoutons l'auteur et la date du document (lignes 93–95).

Nous effectuons d'abord une boucle `for-each` sur les éléments de type `div1` (lignes 96–104). À l'intérieur nous appelons d'abord le modèle nommé `header` (ligne 97), qui génère un numéro de section basé sur les éléments de type `div1` déjà rencontrés (ligne 45) puis extrait la valeur de l'élément enfant `head` de `div1` (ligne 46).

Ensuite, nous commençons une liste non-numérotée (ligne 98) qui contient une entrée pour chaque occurrence d'un élément `div2` (boucle `for-each` en lignes 99–102). Pour un élément de type `div2` donné, nous mettons la valeur de la variable `poemname` (ligne 100) en utilisant la fonction `generate-id()` qui associe un numéro de référence interne *unique* au nœud courant. Cette valeur est utilisée à la ligne 101 dans la référence `href` de l'ancre de type `a` qui lie le fichier principal `verlaine.html` (défini par la variable `MasterFile` en ligne 13) que nous sommes en train d'écrire aux fichiers contenant les poèmes individuels, et dont le nom est construit à partir de cet identificateur unique précédé de la valeur de la variable `MasterFile` (voir l'attribut `select` de la ligne 100).

Après avoir traité tous les éléments `div1` et `div2` nous terminons le fichier principal par un appel au modèle nommé `footer` (ligne 105). Ce modèle est défini en lignes 21–42 et génère quelques balises HTML avec du texte (lignes 22–23 et 40–41) en y insérant la date (voir ci-dessus pour une discussion de l'utilisation des fonctions d'extension).

Finalement, ligne 108, nous initions la génération des fichiers individuels avec les poèmes en sélectionnant le traitement des éléments de type `div2`.

Les fichiers avec les poèmes sont générés dans le modèle qui traite le motif `div2` (lignes 111–128) et dont le traitement est initié en ligne 118.

Nous commençons (ligne 112) par définir une variable `file` qui utilisera la même procédure qu'en ligne 100 pour calculer le nom à donner à chaque nœud de type `div2` traité. La norme XSLT garantit que chaque nœud dans l'arborescence du document complet a un identificateur unique généré à l'aide de la fonction `generate-id`, ainsi nous sommes certains que les noms obtenus en ligne 100 et ici pour nommer les fichiers via la variable `file` seront identiques.

La ligne 113 crée un document HTML ayant un nom donné par la valeur de la variable `file` définie en ligne précédente et utilise la codage UTF-8. La possibilité de créer des fichiers étant une extension propre à `xt`, nous utilisons le domaine nominal `xt:` défini en ligne 8 pour l'élément `xt:document` (lignes 113 et 127).

Le reste de la construction des fichiers contenant un poème est assez facile. La ligne 116 génère un titre pour l'en-tête HTML. La partie la plus intéressante est en ligne 120, où nous faisons appel au modèle nommé `xref`. Elle est définie en lignes 49–78. Elle a un paramètre `home` qui donne le nom de la page principale et sa valeur est définie en ligne 121 en utilisant la valeur de la variable `MasterFile`.

Le modèle nommé `xrefpanel` utilise une table pour mettre à la tête de chaque fichier des liens qui permettent à l'utilisateur de naviguer d'un fichier au précédent, au suivant et de remonter au fichier principal. Cette dernière possibilité est facile à traiter puisque le fichier principal est trivial à adresser (ligne 54). Par contre lorsque nous nous trouvons au début de la séquence `div2` ou à sa fin les tests en lignes 56 et 65 ne permettent plus de trouver le précédent, respectivement, le suivant. En effet les axes `preceding-sibling` et `following-sibling`, qui trouvent des nœuds au même niveau que l'élément courant, ne peuvent sortir de l'élément parent. Il nous faut des tests un peu plus complexes, c'est-à-dire : nous devons sauter d'un élément `div1` à un autre pour s'assurer que nous trouvons bien toujours tous les types d'élément précédent ou suivant. C'est ce que nous faisons dans les lignes 60–61 et 69–70. Par exemple, en ligne 61 nous savons (par le test de la ligne 56) que nous n'avons plus d'élément `div2` précédent l'élément courant à l'intérieur de l'élément `div1` qui est l'ancêtre de l'élément courant. Donc nous devons d'abord monter d'un niveau pour trouver un élément de type `div1` (`ancestor::div1`). Puis nous choisissons parmi les nœuds de type `div1` précèdents le nœud ancêtre le plus proche (`preceding-sibling::div1[1]`). Pour ce nœud nous sélectionnons le dernier de ses enfants de type `div2` (`/div2[last()]`). C'est ce nœud, s'il existe, qui détermine l'adresse du fichier précédent dans le fichier HTML pour l'élément `div2` courant. La même logique (dans la direction opposée `following`) s'applique pour déterminer si un élément suivant existe au-delà de l'ancêtre `div1` de l'élément courant.

Finalement en ligne 77 nous définissons la cible de l'ancre et nous écrivons le titre en sélectionnant la valeur de l'élément `head` à l'intérieur de `div2`.

Nous retournons en ligne 123 qui envoie le texte du poème pour traitement par les autres modèles (lignes 130–148). Finalement chaque fichier aura une partie adresse avec la date, etc. générée à l'aide du modèle nommé `footer` déjà mentionné.

Le résultat de l'application de la feuille de style `htmlsplit.xsl` au fichier source XML `verlainetei.xml` est montré à la figure 12. Nous y montrons les hyperliens présents dans le document principal par des flèches. De plus chacun des sept fichiers avec les poèmes individuels contient des liens vers le poème précédent et suivant et pour remonter au fichier principal. Ces liens sont mis en évidence par des étiquettes `Monter`, `Précédent` et `Suivant` à la première ligne de chaque fichier (sauf aux deux extrémités). Nous avons entré la commande :

```
java com.jclark.xsl.sax.Driver verlainetei.xml htmlsplit.xsl verlaine.html
```

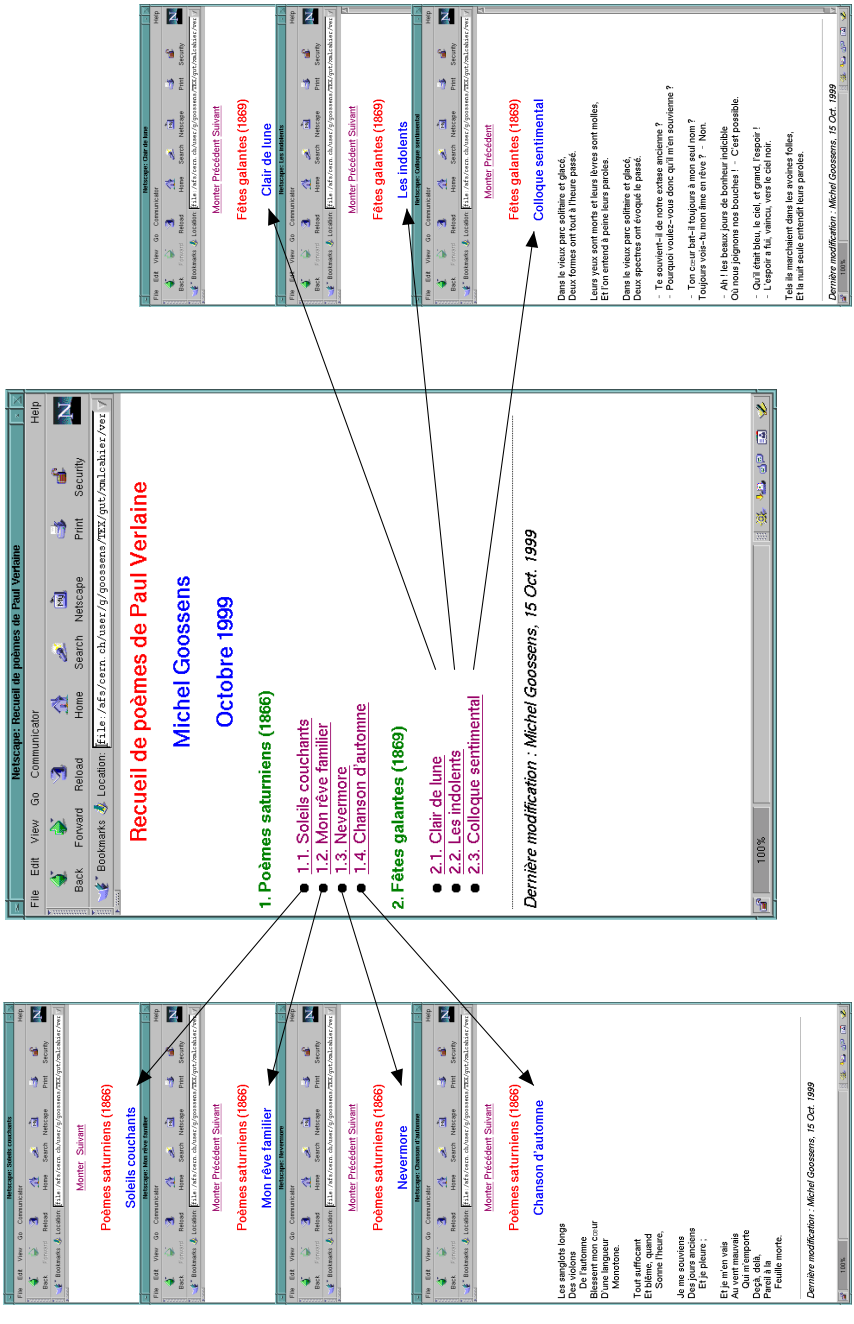


FIGURE 12 – *Une toile de fichiers* HTML, *chacun contenant un poème, généré à partir d'un fichier* XML

7. Autres normes W3C basées sur XML

XML est la norme de base qui permet de définir d'autres applications dans plusieurs domaines d'activité. Parmi les plus intéressantes pour nous, signalons :

- XLink [47] : les liens généralisés ;
- XPointer[49] : l'adressage généralisé ;
- SVG[43] : les graphiques vectoriels ajustables ;
- XHTML [44, 45] : un langage HTML modulaire ;
- XML Schema[50, 51] : une description des structures et types de données.

Nous allons regarder quelques-uns plus de détails. Il faut savoir toutefois qu'aucun de ces travaux n'est actuellement finalisé et ce qui suit ne correspondra peut-être pas à 100% à la syntaxe qui sera finalement adoptée.

7.1. Modèle XLink pour définir des hyperliens

XLink [47] et XPointer[49] proposent une structure généralisée, efficace et compacte pour représenter des liens internes ou externes à un document. XLink se limite à la définition abstraite des *ressources*, unités d'information ou de service adressables par un lien, laissant les détails de présentation aux fichiers de styles ou autres mécanismes semblables.

XLink et XPointer se sont inspirés de trois normes existantes :

HTML : définit plusieurs éléments pour gérer les liens et généralise l'utilisation de la notion d'URL (*Uniform Resource Locator*) qui pointe en majeure partie vers des objets globaux, même si un certain adressage interne est possible (association d'un identificateur à un élément, régions dans des images graphiques) ;

Hytime: (*Hypermedia Time-based Structuring Language*) est une application SGML normalisée par l'ISO[37] qui propose un formalisme standardisé pour exprimer des liens entre textes et objets multimedia (images, son, etc.), pour localiser de façon dynamique dans le temps et l'espace les éléments liés, pour accéder aux propriétés de chaque ressource (« ancre » dans la terminologie de HTML) et pour donner un aperçu de la ressource en question (résumé textuel, extrait sonore, image d'un film, etc.) ;

TEI : (*Text Encoding Initiative*) est une application SGML développée par plusieurs associations des sciences humaines[4, 33]. Dans le domaine de l'hypertexte il définit une syntaxe formelle pour spécifier la location de données structurées, graphiques et autres types de données, et propose des structures pour créer des liens et des collections de liens à partir de ces ressources.

Un *lien* exprime une relation entre des ressources. Une *ressource* peut être n'importe quelle location qui est adressée par le lien. Notons que la nature exacte de la relation entre ces différentes ressources dépend de l'application qui traite le lien ainsi que de l'information sémantique fournie par l'utilisateur.

XLink compte plusieurs types de liens. XPointer, qui est basé sur XPath et s'inspire de l'initiative TEI introduit des méthodes sophistiquées pour localiser les ressources.

Pour permettre aux applications de reconnaître les liens, on définit des éléments et attributs dans le domaine nominal de XLink, par exemple :

```
<A xmlns:xlink="http://www.w3.org/1999/xlink/namespace/" xlink:type="simple"
        xlink:href="pays.xml" xlink:role="liste des pays"
        xlink:title="Liste des pays" xlink:show="new" xlink:actuate="user">
        Liste des pays (à jour au 22 octobre 1999)
</A>
```

En fait, cet élément `A` correspond à l'ancre HTML. Si nous nous trouvons à l'intérieur du domaine nominal `xlink`, nous pouvons déclarer un lien similaire par une définition simplifiée :

```
<xlink:simple href="pays.xml" role="liste des pays"
              title="Liste des pays" show="new" actuate="user">
        Liste des pays (à jour au 22 octobre 1999)
</xlink:simple>
```

Tout élément peut être transformé en un lien en utilisant l'attribut `xlink:type`.

7.1.1. Attributs

XLink permet de caractériser les différentes composantes d'un lien à l'aide d'attributs :

`href` URI identifiant une ressource ;

`from` chaîne de caractères (par exemple un attribut de type `ID`) identifiant la ressource qui a activé le lien ;

`to` chaîne de caractères (par exemple un attribut de type `ID`) identifiant la ressource qui est la destination du lien ; `from` et `to` sont les deux extrémités d'un arc ;

`show` comportement de la ressource pointée ; il existe trois options :

- `new` montrer la ressource pointée dans une nouvelle fenêtre en laissant le contexte actuel intact ;
- `parsed` intégrer le contenu de la ressource pointée dans le document actuel en l'interprétant comme du XML ;
- `replace` remplacer le contexte actuel avec le contenu de la ressource pointée (c'est ce qui s'utilise le plus souvent sur la toile actuellement) ;

`actuate` initialisation du lien ; il existe deux options :
- `user` le lien doit être traversé uniquement quand une requête explicite est reçue par l'application (ce comportement est celui de HTML où vous devez cliquer explicitement sur un lien avec la souris pour l'activer) ;
- `auto` l'application traversera le lien automatiquement quand elle croit que l'utilisateur l'a atteint (par exemple si le pointeur passe par dessus) ;

`role` attribut sémantique qui décrit à l'aide d'une chaîne de caractères son rôle générique (sa « signification », p. ex. `poème`) ; toutes les ressources qui participent à un lien peuvent être caractérisées par leur propre rôle ;

`title` texte décrivant le lien en question.

Cette description permet de mieux comprendre l'exemple du lien simple donné auparavant.

7.1.2. Différents éléments de type `link`

Il existe plusieurs types de liens :

Liens simples

Un lien *simple* ressemble beaucoup aux éléments `A` de HTML ou `XREF` du projet TEI. Un exemple de la définition d'un lien simple et de son utilisation suivent :

```
<!ELEMENT xlink:simple ANY>
<!ATTLIST xlink:simple
          href     CDATA                          #REQUIRED
          role     CDATA                          #IMPLIED
          title    CDATA                          #IMPLIED
          show     (new|parsed|replace)           "replace"
          actuate  (user|auto)                    "user"
>
```

Pour transformer un élément arbitraire en lien simple on déclare :

```
<!ELEMENT listePays ANY>
<!ATTLIST listePays
     xmlns:xlink   CDATA #FIXED "http://www.w3.org/1999/xlink/namespace/"
     xlink:type    (simple|extended|locator|arc)   #FIXED "simple"
     xlink:href    CDATA                           #REQUIRED
     xlink:role    CDATA                           #IMPLIED
     xlink:title   CDATA                           #IMPLIED
     xlink:show    (new|parsed|replace)            "replace"
     xlink:actuate (user|auto)                     "user"
>
```

Le *lien simple* introduit une relation entre deux ressources, le texte contenant le lien en question et sa destination. Pour un lien simple la description de la destination du lien se trouve *toujours* à l'intérieur du document source : il s'agit d'un lien « `in-line` » ou *interne*.

Un exemple d'utilisation serait le suivant :

```
Pour les curieux il est intéressant de consulter
<listePays href="http://www.xxx.yyy/pays.xml"
           title="Liste des pays" role="liste des pays"
           xlink:show="new" xlink:actuate="user">
           la liste des pays (à jour au 22 octobre 1999).
</listePays>
```

Liens étendus

Un lien étendu peut mettre en évidence des relations entre n'importe quel nombre de ressources et, de plus, il peut être spécifié dans un document qui ne coïncide avec aucune des ressources (les liens peuvent être « `out-of-line` » ou *externes*). Ceci permet de générer des liens pour des supports non-modifiables comme les CDROM ou de filtrer des liens à la demande.

Voici une définition explicite d'un lien étendu :

```
<!ELEMENT xlink:extended ((xlink:arc | xlink:locator)*)>
<!ATTLIST xlink:extended
     role             CDATA                          #IMPLIED
     title            CDATA                          #IMPLIED
     showdefault      (new|parsed|replace)           #IMPLIED
     actuatedefault   (user|auto)                    #IMPLIED >
```

Voici une déclaration de lien étendu pour un élément arbitraire :

```
<!ELEMENT listePays ((xlink:arc | xlink:locator)*)>
<!ATTLIST listePays
     xmlns:xlink   CDATA #FIXED "http://www.w3.org/1999/xlink/namespace/"
     xlink:type    (simple|extended|locator|arc)   #FIXED "extended"
     xlink:href    CDATA                           #REQUIRED
     xlink:role    CDATA                           #IMPLIED
     xlink:title   CDATA                           #IMPLIED
     xlink:show    (new|parsed|replace)            "replace"
     xlink:actuate (user|auto)                     "user"
>
```

Voici deux exemples d'utilisation de ces liens étendus. D'abord, une instance explicite d'un lien de ce type :

```
<xlink:extended role="liste des pays" title="Liste des pays"
       showdefault="replace" actuatedefault="user"> ...
</xlink:extended>
```

puis avec un élément déclaré comme un lien étendu :

```
<listePays xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
           xlink:type="extended"
           xlink:role="liste des pays"
           xlink:title="Liste des pays" xlink:showdefault="replace"
           xlink:actuatedefault="user"> ...
</listePays>
```

Éléments de localisation

Un « localisateur » (*locator*) est contenu dans un lien étendu pour permettre la définition des ressources qui participent au lien. La ressource externe est spécifiée à l'aide de l'attribut `href`. Plusieurs éléments de localisation peuvent être associés à un lien. Chaque ressource externe peut avoir sa propre sémantique par rapport au lien et cette sémantique (les attributs `role` et `title`) est spécifiée dans les éléments de localisation.

La déclaration d'un localisateur explicite est la suivante :

```
<!ELEMENT xlink:locator ANY>
<!ATTLIST xlink:locator
   id         ID           #REQUIRED
   href       CDATA        #REQUIRED
   role       CDATA        #IMPLIED
   title      CDATA        #IMPLIED >
```

Pour utiliser un élément quelconque comme élément de localisation, on déclare :

```
<!ELEMENT étudiant ANY>
<!ATTLIST étudiant
   xmlns:xlink   CDATA #FIXED "http://www.w3.org/1999/xlink/namespace/"
   xlink:type    (locator)  #FIXED "locator"
   id            ID         #REQUIRED
   xlink:href    CDATA      #REQUIRED
   xlink:role    CDATA      #IMPLIED
   xlink:title   CDATA      #IMPLIED
>
```

Dans un document, nous pouvons alors écrire :

```
<xlink:locator href="/poèmes/étudiant/PremierEssai" role="poème"
         title="devoir numéro 1"/>
```

Plus simplement, un élément de localisation peut être utilisé comme suit :

```
<étudiant xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
          xlink:href="PremierEssai">Mon premier poème.</foo>
```

Éléments de type `arc`

Un « arc » est contenu dans un lien étendu pour définir comment le traverser. Plusieurs éléments de type `arc` peuvent être associés à un lien. Les autres attributs de ce type d'élément se comportent comme dans les cas précédents.

La déclaration d'un lien explicite de type `arc` suit :

```
<!ELEMENT xlink:arc ANY>
<!ATTLIST xlink:arc
   from     IDREF          #REQUIRED
   to       IDREF          #REQUIRED >
```

Si nous voulons transformer un autre élément en lien étendu de type `arc` nous pouvons écrire :

```
<!ELEMENT monArc ANY>
<!ATTLIST monArc
   xmlns:xlink   CDATA #FIXED "http://www.w3.org/1999/xlink/namespace/"
   xlink:type    (arc) #FIXED "arc"
   xlink:from    IDREF #REQUIRED
   xlink:to      IDREF #REQUIRED
   show          (new|parsed|replace) "replace"
   actuate       (user|auto)          "user" >
```

Voici deux exemples d'utilisation :

```
<xlink:arc from="étudiant" to="poème" show="parsed" actuate="auto"/>

<monArc xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
        xlink:type="arc" xlink:from="étudiant"
        xlink:to="professeur" show="new" actuate="user"/>
```

Groupes de liens étendus

Un *groupe de liens étendus* est un document XML qui peut contenir une série de liens étendus ou d'autres groupes de liens étendus. Cette facilité est intéressante pour accéder à des ressources adressables à l'aide d'une liste de liens stockée dans un autre document.

Dans ce cas, il est souhaitable d'utiliser un élément de type *groupe de liens étendus*, un lien étendu spécial, pour y enregistrer une liste de liens vers d'autres documents qui, ensembles, forment un groupe de documents liés entre eux. Chaque document de ce type est identifié par un élément de type *document de liens étendus*, cas spécial d'élément de localisation.

Des exemples de déclaration de groupe de liens étendus dans un document de liens étendus, montrant les attributs possibles liés à `xlink` sont donnés ci-après. En particulier, la valeur de l'attribut `xlink:type` doit être `group` ou `document`, respectivement.

```
<!ELEMENT xlink:group (xlink:document*)>
<!ATTLIST xlink:group steps CDATA #IMPLIED >

<!ELEMENT xlink:document EMPTY>
<!ATTLIST xlink:document href CDATA #REQUIRED >

<xlink:group steps="2">
    <xlink:document href="http://www.w3.org/W3CHubDoc.xml"/>
</xlink:group>
```

Puisqu'un groupe de liens étendus peut rediriger une application vers un autre document, qui peut à son tour contenir d'autres groupes de liens étendus, il est possible de se trouver dans une boucle infinie. Ainsi l'attribut `steps` de type numérique devrait indiquer combien de niveaux de liens seraient à traiter. L'utilisation de cet attribut fort utile n'est pas obligatoire.

7.2. XPointer ou les pointeurs généralisés

Un localisateur pour une ressource consiste habituellement en une référence URI. Pour adresser un fragment de ressource, on peut utiliser un *pointeur généralisé* de type XPointer.

Pour un document XML, XPointer travaille sur le même arbre qu'XSL et les expressions XPath sont la base de la syntaxe XPointer, qui sélectionne certaines parties de l'arbre, par exemple en indiquant leur relation structurelle par rapport à certains

autres nœuds. Des pointeurs de type XPointer peuvent agir de façon itérative en exprimant des sélections multiples en chaîne, chacune agissant sur la partie de l'arbre sélectionnée au stade précédent.

La sélection se fait à l'aide d'axes et de prédicats (comme avec XPath).

Les identificateurs de fragments XPointer existent sous trois formes : deux courtes et une complète :

- par compatibilité avec HTML, la première forme courte utilise seulement le nom simple d'un élément ;
- la deuxième forme courte localise un élément par navigation par étapes à partir de l'élément racine en utilisant une séquence de nombres séparés par des signes « / » ;
- la forme complète permet d'adresser n'importe quel fragment d'un document XML. Elle se base directement sur le langage XPath qui a été étendu pour permettre la modélisation de sélections utilisateurs et pour s'assurer qu'une pointeur XPointer retourne une seule partie contiguë d'un document.

Les extensions à XPath sont énumérées brièvement ci-dessous.

- La spécification de règles pour initialiser le contexte d'évaluation.
- Deux nouveaux axes pour modeler le choix utilisateur.
  L'axe `range::` adresse une partie d'un document entre un début et une fin, par exemple :

  ```
  "id('b37')/range::child[1],following-sibling[2]"
  ```

  sélectionne la région du document allant du premier au troisième élément enfant de l'élément identifié avec l'`ID` « `b37` »

  ```
  "range::id('section3.1')/p[last()-1],id('section3.3')/p[2]"
  ```

  sélectionne tout à partir de l'avant dernier paragraphe (`p[last()-1]`) de l'élément ayant l'identificateur `section3.1` jusqu'au deuxième paragraphe (`p[2]`) de l'élément ayant l'identificateur `section3.3`.
  L'axe `string::` adresse une partie d'un document à l'aide d'une chaîne de caractères, par exemple si l'élément identifié par l'`ID` « `b37` » contenait la chaîne de caractères « Association GUTenberg » le pointeur XPointer

  ```
  "id('b37')/string::5,''"
  ```

  localisera le point précédant la cinquième lettre (« c »). Avec le pointeur XPointer

  ```
  "string::4,'Association GUTenberg',15,2"
  ```

localisera « Te » dans la troisième occurrence de la chaîne de caractères « Association GUTenberg ». Les balises sont ignorées, c'est-à-dire : si le document contenait la chaîne « Association GUTenberg » le pointeur XPointer

```
"string::1,'ion GUT',-3,3"
```

localisera « GUT ».

– De nouveaux chemins de localisation.
`"/"` désigne la ressource complète.
La fonction `id()` localise l'élément dans la ressource avec un attribut de type `id` et avec la valeur indiquée. Par exemple :

```
"id('b37')"
```

La fonction `here()` permet la localisation du nœud de l'élément qui contient directement un pointeur XPointer (dans le texte ou comme attribut).
La fonction `origin()` permet l'adressage relatif à des liens externes comme ceux définis dans XLink.
– Une nouvelle fonction `unique()` pour les prédicats. Elle retourne la valeur booléenne « vrai » si la liste courante du nœud contexte contient exactement un nœud.

7.3. SVG ou l'adieu à JPEG, GIF et PNG . . . ?

SVG[43] (*Scalable Vector Graphics* ou « graphique vectoriel ajustable») combine les avantages des langages PGML (*Precision Graphics Markup Language* d'Adobe) et VML (*Vector Markup Language* de Microsoft) et propose un seul langage « simple » pour inclure des informations graphiques dans les fichiers XML de façon portable.

SVG peut être utilisé dans des fichiers indépendants pour représenter les dessins. Ceux-ci peuvent être créés par des applications spécialisées et inclus dans les pages web par exemple par une référence XPointer. Par ailleurs on peut également inclure directement des fragments SVG dans les sources XML.

Voici un exemple de fichier SVG, qui dessine divers rectangles et ellipses.

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE svg SYSTEM "svg-19990730.dtd" >
<svg width="20cm" height="175mm">
  <g style="stroke:black; fill:none; text-antialiasing:true;stroke-antialiasing:true" >
    <text x="5" y="25" style="font-size:24">Exemple SVG :
                              rectangles et ellipses</text>
      <rect x="10"  y="50"  width="100" height="75"  />
      <rect x="120" y="50"  width="100" height="75" style="fill:red" />
      <rect x="230" y="50"  width="100" height="75"
                    style="stroke:lime; stroke-width:6" />
      <rect x="340" y="50"  width="100" height="75"
            style="stroke:blue; fill:none; stroke-width:3;
            stroke-dasharray:10 5;stroke-linejoin:miter"  />
    </g>
```

```
15      <g style="stroke:none; fill:blue">
16        <rect x="10"  y="150"  width="100" height="75" rx="40" ry="40" />
17        <rect x="120" y="150"  width="100" height="75" rx="50" ry="50" />
18        <rect x="230" y="150"  width="100" height="75" rx="60" ry="60"
19                      style="stroke:lime; stroke-width:6" />
20        <rect x="340" y="150"  width="100" height="75" rx="70" ry="70"
21              style="stroke:blue; fill:none; stroke-width:3;
22              stroke-dasharray:6 3;stroke-linejoin:miter"  />
23      </g>
24      <ellipse cx="60"  cy="300" rx="50" ry="40"  />
25      <ellipse cx="170" cy="300" rx="50" ry="40" style="fill:red" />
26      <ellipse cx="280" cy="300" rx="50" ry="40"
27               style="stroke:lime; stroke-width:6" />
28      <ellipse cx="390" cy="300" rx="50" ry="40" angle="45"  />
29  </svg>
```

La figure 13 montre le résultat de l'interprétation de ces instructions par un programme de visualisation SVG[19].

Il est certain que le langage SVG va jouer un rôle très important dans le monde internet comme format graphique portable.

7.4. XML Schema : au delà de la DTD

Le but principal de XML est l'échange de données d'une base de données à une autre ou entre une base de données et une autre représentation propre à une autre application, un document électronique étant considéré comme une forme spécifique d'une base de données. La spécification XML Schema devrait faciliter la mise en œuvre de ces échanges.

La DTD informe le destinataire d'un document quels éléments, entités, etc. le document en question peut contenir sous une forme facile à exploiter par un programme informatique. Grâce aux mots-clés et définitions dans la DTD, un programme d'édition peut analyser un document XML pour y repérer certains éléments ou pour indiquer à l'utilisateur si toute l'information marquée comme obligatoire est bien présente.

Les schémas XML vont plus loin que la DTD dont une des limitations majeures est de ne pas être (pour des raisons de compatibilité SGML) sous forme d'un document XML. Aussi, les nombreux outils XML disponibles sur l'internet ne peuvent être utilisés pour analyser, composer ou développer les DTD. Néanmoins, se limiter à réécrire l'information présente dans une DTD en XML ne serait qu'une amélioration mineure. Plus important est la possibilité de pouvoir spécifier le type de données qu'un élément peut contenir ainsi que leurs valeurs possibles[51].

La recommandation XML Schema[50] indique également comment structurer l'information pour combiner des données individuelles en entités ou comment combiner les données venant de plusieurs sources. Elle propose une approche structurelle pour documenter un Schema à l'aide de commentaires structurés, ce qui permettra aux

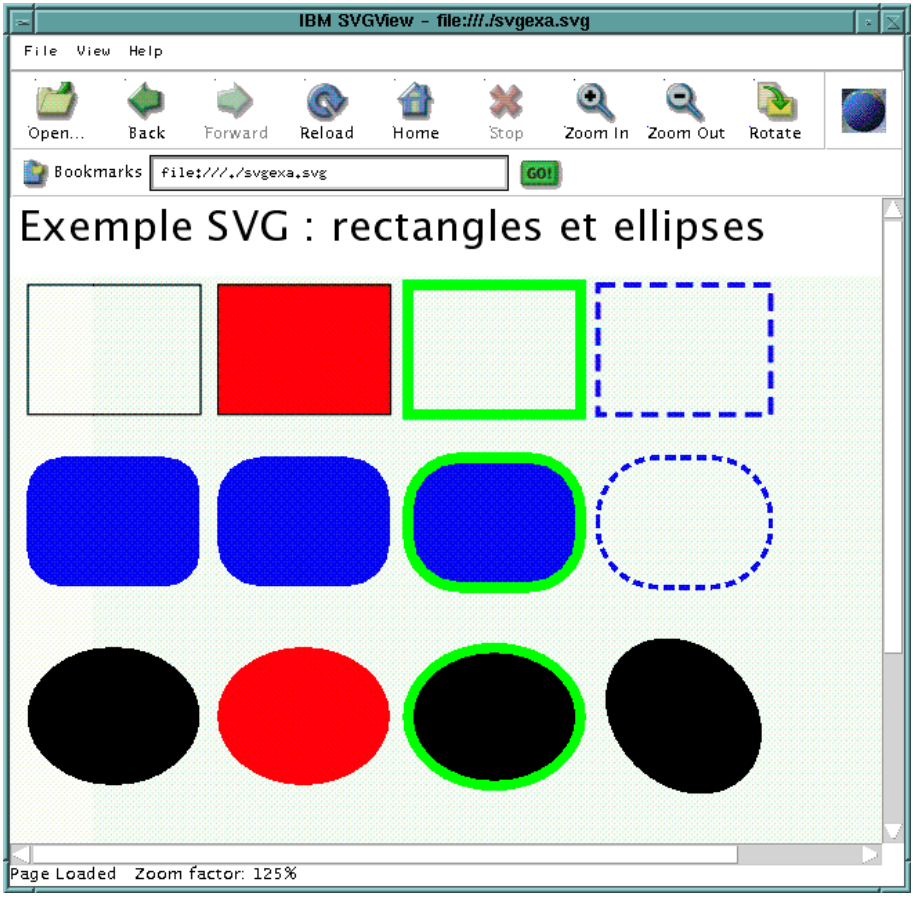


FIGURE 13 – *Quelques formes graphiques simples dessinées avec* SVG

programmes informatiques (p. ex. les éditeurs structurels XML) d'assister l'utilisateur en l'informant des caractéristiques d'un type d'élément. Ces commentaires faciliteront la gestion des différentes versions et mises-à-jour d'un schema. Finalement des principes de conformité garantiront que les schémas venant des différents points du globe pourront fonctionner ensemble.

Conclusions

Dans cet article, nous avons d'abord parlé des problèmes liés au langage HTML. Ensuite, nous avons expliqué pourquoi nous pensons que l'introduction de XML per-

mettra la mise en place d'un environnement sur l'internet véritablement global pour le traitement, l'échange et le stockage de l'information électronique.

Nous avons passé en revue les principales composantes du langage XML en nous basant surtout sur des exemples qui ont mis en évidence comment XML « démocratise » l'internet en permettant à chacun de définir son propre langage grâce aux noms d'attributs et d'éléments faciles à comprendre et à mettre en œuvre. Ainsi les internautes chinois, russes ou français pourront-ils saisir leurs documents en se servant de balises dans leur langue maternelle et être sûrs qu'elles seront adaptées au domaine d'application en question (finance, chimie, biologie, commerce électronique, automobile, etc.).

La deuxième partie de l'article a traité du langage XSL pour transformer les documents XML en plusieurs formats. Nous avons montré comment obtenir une sortie LaTeX, d'abord en transformant les éléments XML directement en commandes LaTeX équivalentes, ou en se servant d'une transformation en objets de formatage XSL, interprétés alors par PassiveTeX, une version spécifique de moteur TeX, ou FOP, générant du PDF directement. Nous avons également abordé le problème de la génération de fichiers HTML et la création d'une toile d'hyperliens reliant les différentes parties entre elles.

Pour ceux qui sont moins intéressés à la présentation de l'information qu'à sa manipulation, nous avons expliqué comment transformer l'information en plusieurs formes plus adaptées aux bases de données.

Nous espérons que, grâce aux exemples détaillés, chacun pourra utiliser ces techniques pour commencer à développer ses propres feuilles de style et ainsi pleinement profiter des avantages de la familles des langages extensibles.

Dans les dernières pages nous avons brièvement décrit quelques autres langages. Les deux premiers, XLink et XPointer, définissent des liens hypertextes beaucoup plus flexibles que ceux disponibles en HTML. SVG deviendra sans doute très vite le format graphique préféré de l'internet. XML Schema définit le contenu structurel et les types de données pour une famille de documents XML. Toutefois, il est important de souligner que la normalisation de ces langages n'est pas aussi avancée que XSL, CSS ou le DOM (voir à la page 155 l'article de François ROLE et Philippe VERDRET).

Annexes

La section A présente une DTD qui s'inspire de LaTeX et qui permet une traduction facilitée de documents saisis d'après cette DTD en LaTeX. De façon plus générale, l'utilisation de cette DTD offre une approche quasi automatique pour sauvegarder des documents LaTeX en XML et ainsi pleinement profiter de tous les outils XML disponibles sur l'internet.

Les sections B à E présentent les règles de productions, instructions et fonctions décrites dans les recommandations des domaines nominaux, XPath et XSLT. Ils permettent de connaître relativement facilement la syntaxe de ces éléments mais pour plus de détails le lecteur doit évidemment se référer aux textes des recommandations en question.

La section F propose une table avec des équivalents français pour certains termes techniques XML.

Pour information et sans commentaires, la section G montre une version de la feuille de style XSL utilisée pour traduire la norme XML en LaTeX.

A. LaTeX et XML

A.1. Une DTD pour LaTeX

Notre intention est de disposer d'une DTD XML qui permette de modéliser relativement facilement des documents initialement balisés en LaTeX. Pour limiter la complexité, nous ne proposons pas une DTD qui englobe toutes les fonctions de LaTeX, y compris celles du processeur mathématique de TeX. Nous nous bornons à évoquer l'approche à suivre avec un problème de ce type.

Ci-dessous, nous montrons la DTD `minilatex.dtd` qui réalise en large partie les commandes et environnements structurels de LaTeX en y ajoutant quelques extensions pour mieux tirer profit des possibilités de XML. Par exemple, nous associons à presque tous les éléments un identificateur (`id` en ligne 3), une classe (`class` en ligne 4), un style (`style` en ligne 5) et un langage (`xml:lang` en ligne 7), le tout encapsulé à l'entité paramètre `basic` (ligne 8).

Pour modulariser la feuille de style, nous définissons des entités paramètres pour chaque type de commande ou environnement (lignes 10–27). Il faut aussi remarquer que nous permettons l'extension de cette DTD par l'utilisateur par l'inclusion d'entrées ayant le suffixe `.new` (p. ex. `fontchange.new` en ligne 10, etc.). En redéfinissant une de ces entités dans le sous-ensemble interne de la DTD, l'utilisateur peut introduire ses propres éléments dans la structure du document. Un exemple de l'utilisation de ce procédé est dans le fichier `minilatexexa.xml` (voir page A.1) où en ligne 9 nous introduisons un nouvel élément `footnote` dans l'entité paramètre `inline.new`. Nous devons aussi déclarer l'élément dans la DTD (voir ligne 10). Il faut évidemment faire attention à ce que le modèle de contenu (la structure de la DTD) reste valable. C'est justement dans ce but que les éléments sont considérés par groupes et traités par des entités différentes. Dans notre cas, l'entité `inline` regroupe des éléments qui peuvent être utilisés à l'intérieur d'un paragraphe (voir la ligne 19 du fichier `minilatex.dtd` ci-dessous). Nous y voyons également comment la partie

extensible (l'entité `inline.new`) est introduite à l'aide d'un appel d'entité paramètre au bout de la déclaration de l'entité paramètre `inline`. Dans le reste du fichier, nous reconnaissons facilement l'équivalent XML des commandes et environnements LaTeX. Nous mentionnons en particulier l'inclusion de graphiques à l'aide du type d'élément `includegraphics`, où les attributs déclarés en lignes 95–100 correspondent directement aux arguments de la commande `\includegraphics` de LaTeX[14, 23]. Pour l'environnement `tabular` de LaTeX, nous déclarons un élément `tabular` qui a un modèle de contenu (lignes 104–116) très proche de la syntaxe LaTeX.

```
1  <!-- minilatex.dtd: XML version of a subset of LaTeX -->
2  <!-- Most elements have the following attributes  -->
3  <!ENTITY % all    "id    ID #IMPLIED
4                     class CDATA #IMPLIED
5                     style CDATA #IMPLIED">
6  <!-- Most non-empty elements have this language attribute -->
7  <!ENTITY % i18n   "xml:lang NMTOKEN #IMPLIED">
8  <!ENTITY % basic  "%all; %i18n;">
9  <!-- Declaration of parameter entities for structuring the DTD -->
10 <!ENTITY % fontchange.new "">
11 <!ENTITY % fontchange "emph|textbf|textsc|textsf|textsl|texttt %fontchange.new;">
12 <!ENTITY % misc.new "">
13 <!ENTITY % misc "url|quad|hspace|vspace|includegraphics|tag|ent %misc.new;">
14 <!ENTITY % xref.new  "">
15 <!ENTITY % xref "cite|pageref|ref %xref.new;">
16 <!ENTITY % mathobj.new "">
17 <!ENTITY % mathobj "displaymath|inlinemath|equation|eqnarray" >
18 <!ENTITY % inline.new "">
19 <!ENTITY % inline  "%fontchange;|quote|tabular|%misc;|%xref;|%mathobj; %inline.new;">
20 <!ENTITY % list.new "">
21 <!ENTITY % list    "lalist|description|enumerate|itemize|bibliography %list.new;">
22 <!ENTITY % preformat.new "">
23 <!ENTITY % preformat "verbatim|verse %preformat.new;">
24 <!ENTITY % likepara.new "">
25 <!ENTITY % likepara "%list;|quotation|align|%preformat; %likepara.new;">
26 <!ENTITY % floats.new "">
27 <!ENTITY % floats "figure|table %floats.new;">
28
29 <!-- The top level document declarations                    -->
30 <!ELEMENT document (frontmatter?,bodymatter,backmatter?)>
31 <!ATTLIST document %i18n;
32                    class CDATA "article">
33 <!ELEMENT frontmatter (title,author?,date?,abstract?)>
34 <!ELEMENT bodymatter  ((par|%likepara;|%floats;|part|chapter|section)*,appendix*)>
35 <!ELEMENT backmatter  (index|glossary)*>
36 <!-- fontchanges -->
37 <!ELEMENT emph    (#PCDATA|%inline;)*>
38 <!ELEMENT textbf  (#PCDATA|%inline;)*>
39 <!ELEMENT textsc  (#PCDATA|%inline;)*>
40 <!ELEMENT textsf  (#PCDATA|%inline;)*>
41 <!ELEMENT textsl  (#PCDATA|%inline;)*>
42 <!ELEMENT texttt  (#PCDATA|%inline;)*>
43 <!-- cross-references -->
44 <!ENTITY % idref  "refid IDREF #REQUIRED">
45 <!ELEMENT cite     EMPTY>
46 <!ATTLIST cite     %idref;>
47 <!ELEMENT pageref  EMPTY>
48 <!ATTLIST pageref  %idref;>
49 <!ELEMENT ref      EMPTY>
50 <!ATTLIST ref      %idref;>
```

```
<!-- quoted material inline and displayed -->
<!ELEMENT quote        (#PCDATA|%inline;|par)*>
<!ATTLIST quote        %basic;>
<!ELEMENT quotation    (#PCDATA|%inline;|par)*>
<!ATTLIST quotation    %basic;>
<!-- lists -->
<!ELEMENT description  ((term*,item*)+)>
<!ATTLIST description  %basic;>
<!ELEMENT enumerate    ((term*,item*)+)>
<!ATTLIST enumerate    %basic;>
<!ELEMENT itemize      ((term*,item*)+)>
<!ATTLIST itemize      %basic;>
<!ELEMENT lalist       ((term*,item*)+)>
<!ATTLIST lalist       %basic;>
<!ELEMENT term         (#PCDATA|%inline;)*>
<!ATTLIST term         %basic;>
<!ELEMENT item         (#PCDATA|%inline;|par|%likepara;)*>
<!ATTLIST item         %basic;>
<!ELEMENT bibliography (bibitem)*>
<!ATTLIST bibliography %basic;>
<!ELEMENT bibitem      (#PCDATA|%inline;|par|%likepara;)*>
<!ATTLIST bibitem      %basic;>
<!-- low-level bits and pieces -->
<!ELEMENT url    EMPTY>
<!ATTLIST url    name CDATA #REQUIRED>
<!ELEMENT quad   EMPTY>
<!ELEMENT hspace EMPTY>
<!ATTLIST hspace dim CDATA #REQUIRED>
<!ELEMENT vspace EMPTY>
<!ATTLIST vspace dim CDATA #REQUIRED>
<!ELEMENT tag    (#PCDATA)>
<!ELEMENT ent    EMPTY>
<!ATTLIST ent    value CDATA #REQUIRED
                 name  CDATA #REQUIRED>
<!-- everything that can go into a paragraph -->
<!ELEMENT par     (#PCDATA|%inline;|%likepara;)*>
<!ATTLIST par     %basic;>
<!-- "floats" and their contents -->
<!ELEMENT figure  (#PCDATA|par|%inline;|%likepara;|caption)*>
<!ATTLIST figure  %basic;>
<!ELEMENT table   (#PCDATA|par|%inline;|%likepara;|caption)*>
<!ATTLIST table   %basic;>
<!ELEMENT includegraphics EMPTY>
<!ATTLIST includegraphics %basic;
                          file     CDATA #REQUIRED
                          width    CDATA #IMPLIED
                          height   CDATA #IMPLIED
                          bb       CDATA #IMPLIED
                          scale    CDATA ".5"
                          angle    CDATA #IMPLIED>
<!ELEMENT caption  (#PCDATA|par|%inline;)*>
<!ATTLIST caption  %basic;>
<!-- tabular material -->
<!ELEMENT tabular (hline|row)*>
<!ATTLIST tabular %basic;
                  preamble CDATA  #REQUIRED
                  width    CDATA  #IMPLIED
                  border   CDATA  #IMPLIED>
<!ELEMENT row     (cell)*>
<!ATTLIST row     %basic;>
<!ELEMENT hline   EMPTY>
<!ELEMENT cell    (#PCDATA|%inline;)*>
<!ATTLIST cell    %basic;
```

```
                    rowspan CDATA "1"
                    colspan CDATA "1"
                    align   (left|center|right) "center">
<!-- verbatim material -->
<!ELEMENT verbatim (#PCDATA)>
<!ATTLIST verbatim %basic;
                    xml:space (default|preserve) 'preserve'>
<!ELEMENT verse    (#PCDATA|%inline;)*>
<!ATTLIST verse    %basic;
                    xml:space (default|preserve) 'preserve'>
<!-- things that go in the frontmatter -->
<!ELEMENT newline  EMPTY>
<!ELEMENT title    (#PCDATA|%inline;|thanks|newline)*>
<!ATTLIST title    %basic;>
<!ELEMENT author   (#PCDATA|%inline;|name|thanks|inst)*>
<!ELEMENT name     (#PCDATA|%inline;)*>
<!ATTLIST name     %basic;>
<!ELEMENT thanks   (#PCDATA|%inline;|newline)*>
<!ATTLIST thanks   %basic;>
<!ELEMENT inst     (#PCDATA|%inline;|newline)*>
<!ATTLIST inst     %basic;>
<!ELEMENT date     (#PCDATA)>
<!ATTLIST date     %basic;>
<!ELEMENT abstract (par+)>
<!ATTLIST abstract %basic;>
<!-- structuring -->
<!ENTITY % sect          "stitle, (par|%likepara;|%floats;)* ">
<!ELEMENT stitle         (#PCDATA|%inline;)*>
<!ATTLIST stitle          %basic;>
<!ELEMENT part           (%sect;, (chapter|section|subsection|subsubsection)*)>
<!ATTLIST part            %basic;>
<!ELEMENT chapter        (%sect;, section*)>
<!ATTLIST chapter         %basic;>
<!ELEMENT section        (%sect;, subsection*)>
<!ATTLIST section         %basic;>
<!ELEMENT subsection     (%sect;, subsubsection*)>
<!ATTLIST subsection      %basic;>
<!ELEMENT subsubsection  (%sect;, paragraph*)>
<!ATTLIST subsubsection   %basic;>
<!ELEMENT paragraph      (%sect;, subparagraph*)>
<!ATTLIST paragraph       %basic;>
<!ELEMENT subparagraph   (%sect;)>
<!ATTLIST subparagraph    %basic;>
<!ELEMENT appendix       EMPTY>
<!-- backmatter: index and glossary -->
<!ELEMENT index (par|%likepara;)*>
<!ATTLIST index %basic;>
<!ELEMENT glossary (par|%likepara;)*>
<!ATTLIST glossagry %basic;>
<!-- LaTeX math constructs -->
<!ENTITY % LaTeXmath  "INCLUDE">
<!ENTITY % latexmath.dtd SYSTEM "latexmath.dtd">
<![ %LaTeXmath; [
%latexmath.dtd;
]]>
<!ENTITY % MathML     "IGNORE">
<!ENTITY % latexmml.dtd SYSTEM "latexmml.dtd">
<![ %MathML; [
%latexmml.dtd;
]]>
<!-- Basic XML entities -->
<!ENTITY lt     "&#60;"> <!-- "<" -->
<!ENTITY gt     "&#62;"> <!-- ">" -->
```

```
<!ENTITY amp    "&#38;#38;"> <!-- "&" -->
<!ENTITY apos   "&#38;#39;"> <!-- "'" -->
<!ENTITY quot   "&#38;#34;"> <!-- '"' -->
```

Nous avons pris soin de ne pas lier cette DTD directement à un balisage des mathématiques en utilisant le formalisme de TEX, mais nous avons laissé la porte ouverte à un balisage en MathML (lignes 162–168 pour LATEX, 169–173 pour MathML). Le choix entre les deux syntaxes est géré par la valeur des entités paramètres `LaTeXmath` (défaut `INCLUDE`, voir ligne 164) et `MathML` (défaut `IGNORE`, voir ligne 169). C'est la valeur de ces deux entités qui décide si les sections conditionnelles qu'elles contrôlent (lignes 166–168 pour `LaTeXmath` et lignes 171–173 pour `MathML`) seront incluses ou pas.

Nous avons défini deux DTD séparées pour traiter ces cas. Voici un premièr fichier DTD `latexmath.dtd` pour exprimer les structures mathématiques à la TEX :

```
<!-- latexmath.dtd +++ inline and display math -->
<!ELEMENT  inlinemath  (#PCDATA)>
<!ATTLIST  inlinemath  %all;>
<!ELEMENT  displaymath (#PCDATA)>
<!ATTLIST  displaymath %all;>
<!ELEMENT  equation    (#PCDATA)>
<!ATTLIST  equation    %all;>
<!ELEMENT  eqnarray    (#PCDATA)>
<!ATTLIST  eqnarray    %all;>
<!ELEMENT  align       (#PCDATA)>
<!ATTLIST  align       %all;>
```

L'approche MathML est encapsulée dans la DTD `latexmml.dtd` suivante :

```
<!-- latexmml.dtd -->
<!ELEMENT equation (math)*>
<!ATTLIST equation %all;>
<!ELEMENT displaymath (math)*>
<!ELEMENT inlinemath  (math)*>
<!ELEMENT subeqn (math)*>
<!ATTLIST subeqn %all;>
<!ELEMENT eqnarray (subeqn)*>
<!ATTLIST eqnarray %all;
                   number (yes|no) "yes">
<!ELEMENT align (%inline;)*>
<!ATTLIST align %all; >

<!-- ISO and MathML DTDs and entities -->
<!--Added Math Symbols: Arrows-->
<!ENTITY % isoamsae.dtd SYSTEM "isoamsae.dtd">
<!--Added Math Symbols: Binary Operators-->
<!ENTITY % isoamsbe.dtd SYSTEM "isoamsbe.dtd">
<!--Added Math Symbols: Delimiters-->
<!ENTITY % isoamsce.dtd SYSTEM "isoamsce.dtd">
<!--Added Math Symbols: Negated Relations-->
<!ENTITY % isoamsne.dtd SYSTEM "isoamsne.dtd">
<!--Added Math Symbols: Ordinary-->
<!ENTITY % isoamsoe.dtd SYSTEM "isoamsoe.dtd">
```

```
<!--Added Math Symbols: Relations-->
<!ENTITY % isoamsre.dtd SYSTEM "isoamsre.dtd">
<!--General Technical-->
<!ENTITY % isoteche.dtd SYSTEM "isoteche.dtd">
<!--Numbers and Currency symbols-->
<!ENTITY % isonume.dtd SYSTEM "isonume.dtd">
<!--MathML Aliases (From ISO PUB,DIA,NUM)-->
<!ENTITY % mmaliase.dtd SYSTEM "mmaliase.dtd">
<!--Greek Symbols-->
<!ENTITY % isogrk3e.dtd SYSTEM "isogrk3e.dtd">
<!--Math Script Font-->
<!ENTITY % isomscre.dtd SYSTEM "isomscre.dtd">
<!--Math Open Face Font-->
<!ENTITY % isomopfe.dtd SYSTEM "isomopfe.dtd">
<!--MathML Entities-->
<!ENTITY % mmlent.dtd SYSTEM "mmlent.dtd">
<!--Main MathML DTD -->
<!ENTITY % mathml.dtd    SYSTEM "mathml.dtd">

%mathml.dtd;
%isoamsae.dtd;
%isoamsbe.dtd;
%isoamsce.dtd;
%isoamsne.dtd;
%isoamsoe.dtd;
%isoamsre.dtd;
%isoteche.dtd;
%isonume.dtd;
%mmaliase.dtd;
%isogrk3e.dtd;
%isomscre.dtd;
%isomopfe.dtd;
%mmlent.dtd;
```

C'est l'utilisateur qui définit s'il veut baliser les parties mathématiques de son document dans l'un ou l'autre de ces deux formalismes en chargeant, au moment de la compilation, la première ou la deuxième DTD.

Regardons maintenant comment nous pouvons utiliser les DTD introduites ci-dessus. Nous présentons un exemple de document multilingue LaTeX (basé en partie sur les exemples de la section 9.2 du *The LaTeX companion*[13]) `minilatexexa.xml`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE document SYSTEM "minilatex.dtd"[
<!ENTITY LaTeX  "\LaTeX{}">
<!ENTITY TeX    "\TeX{}">
<!ENTITY dots   "\dots">
<!ENTITY endash "--">
<!ENTITY emdash "---">
<!ENTITY nbsp   "~">
<!ENTITY % inline.new "|footnote">
<!ELEMENT footnote (#PCDATA|par)*>
]>
<document class="article" xml:lang="en">
<frontmatter>
<title>The &LaTeX; DTD and multiple languages</title>
<author><name>Michel Goossens</name>
<thanks>Partly from an example in <emph>The &LaTeX; Companion</emph>
</thanks></author>
```

```
<date>August 4th, 1998</date>
</frontmatter>
<bodymatter>
<section id="sec-en">
<stitle>The basic principles</stitle>
<par>
This is an example input file. We start in English to show the
principle. You should especially pay attention that we have used
slightly different notation for some of the common &LaTeX; constructs,
such as the dashes, which come in three sizes: an intra-word dash, a
medium dash for number ranges like 1&endash;2, and a punctuation
dash&emdash;like this.  Text can be emphasized as <emph>shown
here</emph>. An ellipsis is made with &dots; Footnotes<footnote>This
is a simple footnote.<par>It can also contain <texttt>par</texttt>
elements.</par></footnote> are tricky constructs, since one must be
careful not to nest them.
</par>
<subsection>
<stitle>Dealing with special characters</stitle>
<par>
XML has a different set of reserved characters than &LaTeX;, in
particular, when you want to use any of the three characters
<texttt><![CDATA[\&]]></texttt>, <texttt><![CDATA[<]]></texttt>,
and <texttt><![CDATA[>]]></texttt>, you should enter them as
<texttt>\&amp;amp;</texttt>, <texttt>\&amp;lt;</texttt>, and
<texttt>\&amp;gt;</texttt>, respectively.
</par>
</subsection>
<subsection id="sec-math">
<stitle>&LaTeX; and mathematical formulae</stitle>
<par>
&LaTeX; and <emph>a fortiori</emph> &TeX; are very good at typesetting
mathematical formulae, like
<inlinemath><![CDATA[x- 3 y + z < 7]]></inlinemath> or
<inlinemath><![CDATA[a_{1} > x^{2n} + y^{2n} > x']]></inlinemath> or
<inlinemath><![CDATA[(A, B) = \sum_{i} a_{i} b_{i}]]></inlinemath>.
Do not forget that for reasons of consistency, if you want to refer to
a variable in one of the formulae, such as the symbol
<inlinemath>x</inlinemath>, you must also use math mode in the text.
</par>
</subsection>
</section>
<section xml:lang="de">
<stitle>Beispiel eines Textes in deutscher Sprache</stitle>
<subsection>
<stitle>Eine EPS Abbildung</stitle>
<par>
Dieser Abschnitt zeigt, wie man eine PostScript-Abbildung
<cite refid="bib-PS"/> in ein Dokument einbinden kann.
Abbildung <ref refid="fig-psfig"/> wurde mit dem Befehl
<verbatim>
<![CDATA[\includegraphics[width="3cm"]{file="colorcir.eps}]]>
</verbatim>
in den Text aufgenommen.
</par>
<figure id="fig-psfig">
<includegraphics width="3cm" file="colorcir"/>
<caption xml:lang="de">Ein EPS Bild</caption>
</figure>
</subsection>
<subsection>
<stitle>Beispiel einer Tabelle</stitle>
<par>Die Tabelle <ref refid="tab-exag"/> auf Seite <pageref
```

```
refid="tab-exag"/> zeigt eine Tabelle.
</par>
<table id="tab-exag">
<caption xml:lang="de">
Eingabe der deutschen Zusatzzeichen in &LaTeX;</caption>
<tabular preamble="ccccccc">
<row>
<cell><texttt>&#34;a</texttt> ä</cell>
<cell><texttt>&#34;A</texttt> Ä</cell>
<cell><texttt>&#34;o</texttt> ö</cell>
<cell><texttt>&#34;O</texttt> Ö</cell>
<cell><texttt>&#34;u</texttt> ü</cell>
<cell><texttt>&#34;U</texttt> Ü</cell>
<cell><texttt>&#34;s</texttt> &#223;</cell>
</row>
</tabular>
</table>
</subsection>
</section>
<section xml:lang="fr">
<stitle>Continuation du texte en français</stitle>
<subsection id="sec-list">
<stitle>Traiter les listes</stitle>
<par>
Les listes sont utilisées fréquemment pour structurer ou mettre
en évidence certains éléments d'un document (voir <cite refid="bib-Liste"/>).
</par>
<itemize>
  <item>Ceci est le premier élément d'une liste non-ordonnée. Chaque élément
    de ce type de liste est précédé d'un signe distinctif, comme une
    puce, un tiret, etc.
  </item>
  <item>Ce second élément de la même liste contient une liste de
    <emph>description</emph> imbriquée.
    <description>
      <term>XML</term>
      <item>Meta langage pour définir des classes de documents</item>
      <term>XLL</term>
      <item>Langage pour définir des hyperliens entre différentes
      parties de documents XML</item>
    </description>
    Nous continuons notre texte à l'intérieur de la première liste.
  </item>
</itemize>
</subsection>
</section>
<bibliography>
<bibitem id="bib-PS" xml:lang="de">
Adobe Inc. <emph>PostScript Handbuch (2. Auflage)</emph>
           Addison-Wesley (Deutschland) GmbH, Bonn, 1991.
</bibitem>
<bibitem id="bib-Liste" xml:lang="fr">
Michel Goossens. Personnaliser les listes &LaTeX;.
           <emph>Cahiers GUTenberg</emph>, 17:32&endash;48, mai 1994.
</bibitem>
</bibliography>
</bodymatter>
</document>
```

Dans ce fichier source, nous remarquons en particulier qu'implicitement nous utilisons la notation TEX pour les mathématiques (rappelons que c'est le choix par défaut

de la ligne 164 dans la DTD `minilatex.dtd`). Pour les mathématiques, nous incluons des commandes TEX directement à l'intérieur de sections `CDATA` (lignes 51–53). Les caractères spéciaux de XML sont aussi saisis à l'aide de sections `CDATA` (lignes 40–41), tout comme le contenu du verbatim en ligne 69. En général, lorsqu'on veut inclure du matériel non-XML dans un fichier XML — par exemple des sources en langues de programmation C, C++, Java, etc., qui utilisent les caractères spéciaux de XML, comme `&`, `<` —, il est toujours préférable de le mettre à l'intérieur d'une section `CDATA`.

Les différentes langues sont déclarées à l'aide de l'attribut `xml:lang` sur les éléments `document` (ligne 12 pour l'anglais) et `section` (ligne 60 pour l'allemand et 100 pour le français), ainsi que sur `bibitem` (lignes 128, 132). Pour valider un document XML, cet attribut doit être déclaré pour tout élément auquel il peut être appliqué[3].

Voici une feuille de style XSL `minilatex.xsl` qui traduit ce fichier en LaTeX :

```
1  <?xml version='1.0'?>
2  <!-- minilatex.xsl -->
3  <xsl:stylesheet version="1.0"
4                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
5                  xmlns="http://www.tug.org/latex">
6
7  <xsl:output method="text" indent="no" encoding="ISO-8859-1"/>
8
9  <xsl:strip-space elements="*"/>
10
11 <xsl:template name="label">
12   <xsl:apply-templates/>
13   <xsl:if test="../@id"><xsl:text>\label{</xsl:text>
14   <xsl:value-of select="../@id"/><xsl:text>}</xsl:text></xsl:if>
15 <xsl:text>}
16 </xsl:text>
17 </xsl:template>
18
19 <xsl:template match="document">
20 <xsl:text>\documentclass[]{</xsl:text>
21 <xsl:value-of select="@class"/>
22 <xsl:text>}
23 \usepackage[dvips]{graphicx}
24 \usepackage[T1]{fontenc}
25 \begin{document}
26 </xsl:text>
27 <xsl:apply-templates/>
28 <xsl:text>\end{document}
29 </xsl:text>
30 </xsl:template>
31
32 <!--   ================ Frontmatter element ====================== -->
33 <xsl:template match="frontmatter">
34   <xsl:apply-templates/>
35 <xsl:text>
36 \maketitle
37 </xsl:text>
```

3. Dans la DTD `minilatex.dtd` nous definissons (ligne 7) l'entité paramètre `i18n` contenant `xml:lang`. Puis l'entité paramètre `i18n` est incluse, ligne 8, dans l'entité paramètre `basic`. Celle-ci est associée dans la DTD à pratiquement tous les éléments (voir lignes 53, 55, 58, 60, etc.).

```
</xsl:template>
<xsl:template match="frontmatter/title">
<xsl:text>
\title{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="frontmatter/author">
<xsl:text>
\author{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}
</xsl:text>
</xsl:template>
<xsl:template match="frontmatter/author/name">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="frontmatter/author/thanks">
<xsl:text>
\thanks{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="frontmatter/author/inst">
<xsl:text>
\thanks{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="frontmatter/date">
<xsl:text>
\date{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}
</xsl:text>
</xsl:template>
<xsl:template match="frontmatter/abstract">
<xsl:text>
\begin{abstract}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>
\end{abstract}
</xsl:text>
</xsl:template>
<xsl:template match="frontmatter/keywords">
<xsl:text>\keywords{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>

<!--   ===================== Bodymatter element ================== -->
<xsl:template match="bodymatter|part|chapter|section|subsection|
                     subsubsection|paragraph|subparagraph">
  <xsl:apply-templates/>
</xsl:template>
<!--   ================== Section headings ====================== -->
<xsl:template match="part/stitle">
  <xsl:text>\part{</xsl:text><xsl:call-template name="label"/>
</xsl:template>
<xsl:template match="chapter/stitle">
  <xsl:text>\chapter{</xsl:text><xsl:call-template name="label"/>
</xsl:template>
```

```
<xsl:template match="section/stitle">
  <xsl:text>\section{</xsl:text><xsl:call-template name="label"/>
</xsl:template>
<xsl:template match="subsection/stitle">
  <xsl:text>\subsection{</xsl:text><xsl:call-template name="label"/>
</xsl:template>
<xsl:template match="subsubsection/stitle">
  <xsl:text>\subsubsection{</xsl:text><xsl:call-template name="label"/>
</xsl:template>
<xsl:template match="paragraph/stitle">
  <xsl:text>\paragraph{</xsl:text><xsl:call-template name="label"/>
</xsl:template>
<xsl:template match="subparagraph/stitle">
  <xsl:text>\subparagraph{</xsl:text><xsl:call-template name="label"/>
</xsl:template>
<!--   ==================== Font changes ======================== -->
<xsl:template match="emph">
<xsl:text>\emph{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="textbf">
<xsl:text>\textbf{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="textsc">
<xsl:text>\textsc{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="textsf">
<xsl:text>\textsf{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="textsl">
<xsl:text>\textsl{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="texttt">
<xsl:text>\texttt{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<!--   ================== Cross-references ====================== -->
<xsl:template match="cite">
  <xsl:text>\cite{</xsl:text>
  <xsl:value-of select="@refid"/>
  <xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="pageref">
  <xsl:text>\pageref{</xsl:text>
  <xsl:value-of select="@refid"/>
  <xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="ref">
  <xsl:text>\ref{</xsl:text>
  <xsl:value-of select="@refid"/>
  <xsl:text>}</xsl:text>
</xsl:template>
<!--   =============== quotes, footnotes, verbatim ================ -->
```

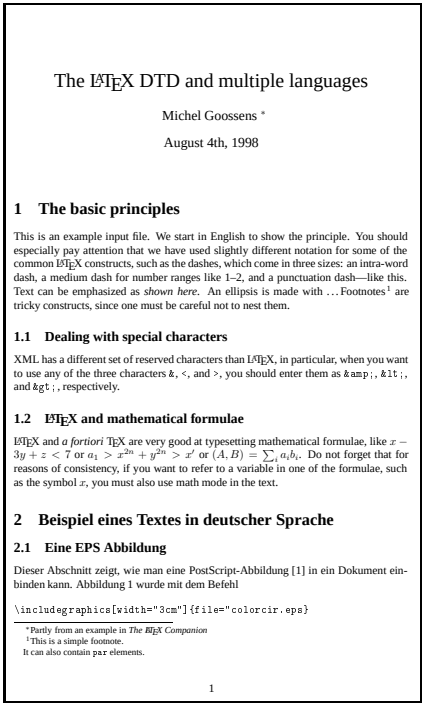
```
<xsl:template match="footnote">
<xsl:text>\footnote{</xsl:text>
  <xsl:apply-templates/>
<xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="quote">
<xsl:text>
\begin{quote}</xsl:text>
  <xsl:apply-templates/>
<xsl:text>\end{quote}</xsl:text>
</xsl:template>
<xsl:template match="quotation">
<xsl:text>
\begin{quotation}</xsl:text>
  <xsl:apply-templates/>
<xsl:text>\end{quotation}</xsl:text>
</xsl:template>
<xsl:template match="verbatim">
<xsl:text>
\begin{verbatim}</xsl:text>
  <xsl:apply-templates/>
<xsl:text>\end{verbatim}</xsl:text>
</xsl:template>
<!--   ======================== Lists ========================== -->
<xsl:template match="description">
<xsl:text>
\begin{description}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>
\end{description}
</xsl:text>
</xsl:template>
<xsl:template match="description/term">
<xsl:text>
\item[</xsl:text>
  <xsl:apply-templates/>
<xsl:text>]</xsl:text>
</xsl:template>
<xsl:template match="description/item">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="enumerate">
<xsl:text>
\begin{enumerate}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>\end{enumerate}
</xsl:text>
</xsl:template>
<xsl:template match="itemize">
<xsl:text>
\begin{itemize}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>\end{itemize}
</xsl:text>
</xsl:template>
<xsl:template match="enumerate|itemize/item">
<xsl:text>
\item </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="bibliography">
<xsl:text>
\begin{thebibliography}{99}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>
\end{thebibliography}
</xsl:text>
</xsl:template>
<xsl:template match="bibliography/bibitem">
<xsl:text>\bibitem{</xsl:text>
<xsl:value-of select="@id"/>
<xsl:text>}</xsl:text>
<xsl:apply-templates/>
</xsl:template>
<!--   ====================== Mathematics ======================== -->
<xsl:template match="inlinemath">
<xsl:text>$</xsl:text>
  <xsl:apply-templates/>
<xsl:text>$</xsl:text>
</xsl:template>
<xsl:template match="displaymath">
<xsl:text>
\begin{displaymath}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>
\end{displaymath}
</xsl:text>
</xsl:template>
<xsl:template match="equation">
<xsl:text>
\begin{equation}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>
\end{equation}
</xsl:text>
</xsl:template>
<xsl:template match="eqnarray">
<xsl:text>
\begin{eqnarray}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>
\end{eqnarray}
</xsl:text>
</xsl:template>
<!--   ====================== A paragraph ======================== -->
<xsl:template match="par">
<xsl:text>
\par
</xsl:text>
  <xsl:apply-templates/>
</xsl:template>
<!--   ======================= Tabular ========================== -->
<xsl:template match="tabular">
<xsl:text>
\begin{tabular}{</xsl:text>
  <xsl:value-of select="@preamble"/><xsl:text>}
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>
```

```
\end{tabular}
</xsl:text>
</xsl:template>
<xsl:template match="tabular/row">
<xsl:apply-templates/>
<xsl:text>\\
</xsl:text>
</xsl:template>
<xsl:template match="tabular/row/cell">
<xsl:apply-templates/><xsl:text>&amp;</xsl:text>
</xsl:template>
<xsl:template match="tabular/row/cell[position()=last()]" priority="2">
<xsl:apply-templates/>
</xsl:template>
<!--   ================== "floats" and their contents ============= -->
<xsl:template match="figure">
<xsl:text>
\begin{figure}\centering
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>\end{figure}
</xsl:text>
</xsl:template>
<xsl:template match="table">
<xsl:text>
\begin{table}\centering
</xsl:text>
  <xsl:apply-templates/>
<xsl:text>\end{table}
</xsl:text>
</xsl:template>
<xsl:template match="figure/caption | table/caption">
  <xsl:text>\caption{</xsl:text><xsl:call-template name="label"/>
</xsl:template>
<!--   =================== Includegraphics ====================== -->
<xsl:template match="includegraphics">
<xsl:text>
\includegraphics[</xsl:text>
<xsl:if test="@width"><xsl:text>width=</xsl:text>
    <xsl:value-of select="@width"/><xsl:text>, </xsl:text></xsl:if>
<xsl:if test="@height"><xsl:text>height=</xsl:text>
    <xsl:value-of select="@height"/><xsl:text>, </xsl:text></xsl:if>
<xsl:if test="@bb"><xsl:text>bb="</xsl:text>
    <xsl:value-of select="@bb"/><xsl:text>, </xsl:text></xsl:if>
<xsl:if test="@angle"><xsl:text>angle=</xsl:text>
    <xsl:value-of select="@angle"/><xsl:text>, </xsl:text></xsl:if>
<xsl:if test="@scale"><xsl:text>scale=</xsl:text>
    <xsl:value-of select="@scale"/><xsl:text></xsl:text></xsl:if>
<xsl:text>]{</xsl:text><xsl:value-of select="@file"/><xsl:text>}
</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Ce fichier de style a une structure assez linéaire et il ne devrait pas y avoir de difficultés particulières à comprendre comment les différents éléments sont traduits en LaTeX. Nous mentionnons seulement la façon dont nous traitons les débuts de chapitres ou de sections (lignes 95–115) où nous avons paramétré la génération du titre et d'une commande LaTeX `\label` à l'aide du modèle nommé `label`, défini en lignes



The LaTeX DTD and multiple languages

Michel Goossens *

August 4th, 1998

1 The basic principles

This is an example input file. We start in English to show the principle. You should especially pay attention that we have used slightly different notation for some of the common LaTeX constructs, such as the dashes, which come in three sizes: an intra-word dash, a medium dash for number ranges like 1–2, and a punctuation dash—like this. Text can be emphasized as *shown here*. An ellipsis is made with ...Footnotes[1] are tricky constructs, since one must be careful not to nest them.

1.1 Dealing with special characters

XML has a different set of reserved characters than LaTeX, in particular, when you want to use any of the three characters &, <, and >, you should enter them as &, <, and >, respectively.

1.2 LaTeX and mathematical formulae

LaTeX and *a fortiori* TeX are very good at typesetting mathematical formulae, like $x - 3y + z < 7$ or $a_1 > x^{2n} + y^{2n} > x'$ or $(A, B) = \sum_i a_i b_i$. Do not forget that for reasons of consistency, if you want to refer to a variable in one of the formulae, such as the symbol x, you must also use math mode in the text.

2 Beispiel eines Textes in deutscher Sprache

2.1 Eine EPS Abbildung

Dieser Abschnitt zeigt, wie man eine PostScript-Abbildung [1] in ein Dokument einbinden kann. Abbildung 1 wurde mit dem Befehl

```
\includegraphics[width="3cm"]{file="colorcir.eps}
```

*Partly from an example in *The LaTeX Companion*

[1] This is a simple footnote.
It can also contain par elements.

1

Figure 1: Ein EPS Bild

Table 1: Eingabe der deutschen Zusatzzeichen in LaTeX

"a	ä	"A	Ä	"o	ö	"O	Ö
"u	ü	"U	Ü	"s	SS		

in den Text aufgenommen.

2.2 Beispiel einer Tabelle

Die Tabelle 1 auf Seite 2 zeigt eine Tabelle.

3 Continuation du texte en français

3.1 Traiter les listes

Les listes sont utilisées fréquemment pour structurer ou mettre en évidence certains éléments d'un document (voir [2]).

- Ceci est le premier élément d'une liste non-ordonnée. Chaque élément de ce type de liste est précédé d'un signe distinctif, comme une puce, un tiret, etc.
- Ce second élément de la même liste contient une liste de *description* imbriquée.

  XML Meta langage pour définir des classes de documents

  XLL Langage pour définir des hyperliens entre différentes parties de documents XML

  Nous continuons notre texte à l'intérieur de la première liste.

References

[1] Adobe Inc. *PostScript Handbuch (2. Auflage)* Addison-Wesley (Deutschland) GmbH, Bonn, 1991.

[2] Michel Goossens. Personnaliser les listes LaTeX. *Cahiers GUTenberg*, 17:32–48, mai 1994.

2

FIGURE 14 – *Un document* XML *balisé avec la* DTD *`minilatex.dtd`*

.

11–17. Nous commençons par écrire le titre (élément `stitle` dans les motifs des lignes 95, 98, 101, etc.) avec l'instruction `<xsl:apply-templates>` de la ligne 12, puis nous vérifions si la section (le parent de `stitle`) a un attribut de type `id` (test `../@id` en ligne 13), auquel cas nous introduisons une commande `\label` ayant comme clé la valeur de l'attribut en question (partie `<xsl:value-of` en ligne 14), puis nous fermons l'accolade contenant le titre de section (ligne 15) dont l'accolade ouverte se trouve en lignes 96, 99, 102, etc.

La figure 14 montre le résultat après traitement du fichier XML `minilatexexa.xml` et la feuille de style XSL `minilatex.xsl` avec un processeur XSL et avec LaTeX.

B. Définition formelle des domaines nominaux

Cette annexe reprend les règles de production de la spécification des domaines nominaux. Le numéro de section dans la spécification[41] où les règles mentionnées

sont introduites est indiqué dans le titre. La notation $^{\mathbb{X}}$ renvoie à un numéro de production dans la spécification XML (voir page 191), $^{\mathbb{N}}$ renvoie à un numéro de production pour les domaines nominaux.

B.1. Déclarer un domaine nominal (section 2)

```
[1]       NSAttName ::= PrefixedAttName^N2 | DefaultAttName^N3
[2] PrefixedAttName ::= 'xmlns:' NCName^N4
[3]  DefaultAttName ::= 'xmlns'
[4]           NCName ::= (Letter^X84 | '_') (NCNameChar^N5)*
[5]      NCNameChar ::= Letter^X84 | Digit^X88 | '.' | '-' | '_' |
                        CombiningChar^X87  | Extender^X89
```

Les productions $^{\mathbb{X}4}$ et $^{\mathbb{X}5}$ de la spécification XML sont remplacées par $^{\mathbb{N}5}$ et $^{\mathbb{N}4}$, respectivement[4].

B.2. Définir les noms qualifiés (section 3)

Le double point est utilisé pour séparer le préfixe (`[7] Prefix`) identifiant le domaine nominal de la partie « locale » (`[8] LocalPart`) du « nom qualifié » (`[8] QNAME`).

```
[6]           QName ::= (Prefix^N7 ':')? LocalPart^N8
[7]          Prefix ::= NCName^N4
[8]       LocalPart ::= NCName^N4
```

B.3. Utiliser les noms qualifiés (section 4)

Avec l'introduction des domaines nominaux, plusieurs productions XML sont généralisées (à gauche le numéro de la production originale dans la spécification de XML).

Règles de production pour les types d'élément

```
[9]^X40          STag ::= '<' QName^N6 (S^X3 Attribute^N12)* S^X3? '>'
[10]^X42         ETag ::= '</' QName^N6 S^X3? '>'
[11]^X44 EmptyElemTag ::= '<' QName^N6 (S^X3 Attribut^N12e)* S^X3? '/>'
```

Règles de production pour les attributs

```
[12]^X41    Attribute ::= NSAttName^N1 Eq^X25 AttValue^X10 |
                          QName^N6 Eq^X25 AttValue^X10
```

4. Les abréviations `NC` et `NS` représentent *non-colon* (sans deux points « : ») et *namespace* (domaine nominal).

Règles de production pour les déclarations

```
[13]X28  doctypedecl ::= '<!DOCTYPE' SX3 QNameN6
                         (SX3 ExternalIDX75)? SX3?
                         ('[' (markupdeclX29 |
                           PEReferenceX69 | SX3)* ']' SX3?)? '>'
[14]X45  elementdecl ::= '<!ELEMENT' SX3 QNameN6 SX3 contentspecX46
                         SX3? '>'
[15]X48          cp ::= (QNameN6 | choiceX49 | seqX50) ('?'|'*'|'+')?
[16]X51       Mixed ::= '(' SX3? '#PCDATA' (SX3? '|' SX3? QNameN6)*
                         SX3? ')*' | '(' SX3? '#PCDATA' SX3? ')'
[17]X52  AttlistDecl ::= '<!ATTLIST' SX3 QNameN6 AttDefN18* SX3? '>'
[18]X53      AttDef ::= SX3 (QNameN6 | NSAttNameN1) SX3 AttTypeX54
                         SX3 DefaultDeclX60
```

C. Définition formelle de XPath

Pour permettre de reconstruire la grammaire de XPath, nous reproduisons dans cette section les différentes règles de production de ce langage et listons également les fonctions disponibles. La notation $^{\mathbb{X}}$ renvoie à un numéro de production dans la spécification XML (voir page 191), $^{\mathbb{N}}$ renvoie à un numéro de production dans la spécification pour les domaines nominaux (voir page 102) et $^{\mathbb{P}}$ à un numéro de production dans Xpath.

C.1. Chemins de localisation (section 2)

Un chemin de localisation (*location path*)

- sélectionne un ensemble de nœuds dans l'arbre source ;
- peut spécifier un chemin absolu commençant par / (relatif au nœud racine du document) ;
- peut spécifier un chemin relatif (ne commençant pas par /) au nœud contexte.

Un chemin de localisation peut contenir plusieurs expressions récursivement sous forme d'une succession de chemins relatifs, collés ensemble avec le signe /.

```
[1]   LocationPath           ::=   RelativeLocationPathP3
                                 | AbsoluteLocationPathP2
[2]   AbsoluteLocationPath   ::=   '/' RelativeLocationPathP3?
                                 | AbbreviatedAbsoluteLocationPathP10
[3]   RelativeLocationPath   ::=   StepP4
                                 | RelativeLocationPathP3 '/' StepP4
                                 | AbbreviatedRelativeLocationPathP11
```

Pas de localisation (section 2.1)

Les pas de localisation (*location step*) sont définit comme suit :

```
[4]   Step                ::=   AxisSpecifier^P5 NodeTest^P7
                                Predicate^P8*
                                | AbbreviatedStep^P12
[5]   AxisSpecifier       ::=   AxisName^P6 '::'
                                | AbbreviatedAxisSpecifier^P13
```

Chaque pas (`Step`$^{\mathbb{P}4}$) dans la construction d'un chemin de localisation se compose :

- d'un identificateur d'axe$^{\mathbb{T}5}$: choisir une direction relative ;
- d'un test sur le nœud$^{\mathbb{T}6}$: que cherche-t-on ?
- de zéro ou plus de prédicats$^{\mathbb{T}7}$: liste d'attributs à utiliser pour sélectionner le(s) nœud(s) en question.

Axes (section 2.2)

```
[6]   AxisName    ::=   'ancestor'          | 'ancestor-or-self'
                      | 'attribute'         | 'child'
                      | 'descendant'        | 'descendant-or-self'
                      | 'following'         | 'following-sibling'
                      | 'namespace'         | 'parent'
                      | 'preceding'         | 'preceding-sibling'
                      | 'self'
```

Tests sur les nœuds (section 2.3)

```
[7]   NodeTest    ::=   NameTest^P37
                        | NodeType^P38 '(' ')'
                        | 'processing-instruction' '(' Literal^P29 ')'
```

Prédicats (section 2.4)

```
[8] Predicate       ::=   '[' PredicateExpr^P9 ']'
[9] PredicateExpr   ::=   Expr^P14
```

Syntaxe courte (section 2.5)

```
[10] AbbreviatedAbsoluteLocationPath ::= '//' RelativeLocationPath^P3
[11] AbbreviatedRelativeLocationPath ::= RelativeLocationPath^P3 '//' Step^P4
[12] AbbreviatedStep                 ::= '.' | '..'
[13] AbbreviatedAxisSpecifier        ::= '@'?
```

C.2. Expressions

Expressions de base (section 3.1)

```
[14] Expr            ::= OrExpr^P21
[15] PrimaryExpr     ::= VariableReference^P36
                       | '(' Expr^P14 ')'
                       | Literal^P29
                       | Number^P30
                       | FunctionCall^P16
```

Appels de fonctions (section 3.2)

```
[16] FunctionCall    ::= FuncionName^P35
                         '(' ( Argument^P17 ( ','Argument )* )? ')'
[17] Argument        ::= Expr^P14
```

Ensembles de nœuds (section 3.3)

```
[18] UnionExpr          ::= PathExpr^P18
                          | UnionExpr^P17 '|' PathExpr^P18
[19] PathExpr           ::= LocationPath^P1
                          | FilterExpr^P19
                          | FilterExpr^P19 '/'
                            RelativeLocationPath^P3
                          | FilterExpr^P19 '//'
                            RelativeLocationPath^P3
[20] FilterExpr         ::= PrimaryExpr^P14
                          | FilterExpr^P17 Predicate^P7
```

Expressions booléennes (section 3.4)

```
[21] OrExpr             ::= AndExpr^P22
                          | OrExpr^P21 'or' AndExpr^P22
[22] AndExpr            ::= EqualityExpr^P23
                          | AndExpr^P22 'and' EqualityExpr^P23
[23] EqualityExpr       ::= RelationalExpr^P24
                          | EqualityExpr^P23 '=' RelationalExpr^P24
                          | EqualityExpr^P23 '!=' RelationalExpr^P24
[24] RelationalExpr     ::= AdditiveExpr^P25
                          | RelationalExpr^P24 '<' AdditiveExpr^P25
                          | RelationalExpr^P24 '>' AdditiveExpr^P25
                          | RelationalExpr^P24 '<=' AdditiveExpr^P25
                          | RelationalExpr^P24 '>=' AdditiveExpr^P25
```

Expressions numériques (section 3.5)

```
[25] AdditiveExpr       ::= MultiplicativeExpr^P26
                            | AdditiveExpr^P25 '+' MultiplicativeExpr^P2
                            | AdditiveExpr^P25 '-' MultiplicativeExpr^P26
[26] MultiplicativeExpr ::= UnaryExpr^P27
                            | MultiplicativeExpr^P26 MultiplyOperator^P34
                            UnaryExpr^P27
                            | MultiplicativeExpr^P26 'div' UnaryExpr^P27
                            | MultiplicativeExpr^P26 'mod' UnaryExpr^P27
[27] UnaryExpr          ::= UnionExpr^P18
                            | '-' UnaryExpr^P27
```

Expressions de structure lexicale (section 3.7)

```
[28] ExprToken          ::= '(' | ')' | '[' | ']' | '.' | '..' |
                            '@' | ',' | '::'
                            | NameTest^P37       | NodeType^P38
                            | Operator^P32       | FunctionName^P35
                            | AxisName^P6        | Literal^P29
                            | Number^P30         | VariableReference^P36
[29] Literal            ::= '"' [^"]* '"'           | "'" [^']* "'"
[30] Number             ::= Digits^P31 ('.' Digits^P31?)?
                            | '.' Digits^P31
[31] Digits             ::= [0-9]+
[32] Operator           ::= OperatorName^P33    | MultiplyOperator^P34
                            | '/' | '//' | '|'  | '+' | '-' | '=' |
                             '!=' | '<'  | '<=' | '>' | '>='
[33] OperatorName       ::= 'and' | 'or' | 'mod' | 'div'
[34] MultiplyOperator   ::= '*'
[35] FunctionName       ::= QName^N6 - NodeType^P38
[36] VariableReference  ::= '$' NCName^N4
[37] NameTest           ::= '*' | NCName^N4 ':' '*' | QName^N6
[38] NodeType           ::= 'comment' | 'node'
                            | 'processing-instruction' |  'text'
[39] ExprWhitespace     ::= S^X3
```

D. Définition formelle de XSLT

La recommandation XSLT utilise le même modèle pour les données et les expressions que XPath, qui a été défini dans la section précédente. La notation $^{\mathbb{P}}$ renvoie à un numéro de production pour la recommandation XPath (voir page 104) et $^{\mathbb{T}}$ à un numéro de production dans XSLT.

D.1. Motifs (XSLT section 5.2)

Les motifs de sélection de XSLT sont un sous-ensemble de ceux disponibles avec XPath. Comme montré ci-dessous, seuls les axes `child` et `attribute` peuvent être directement utilisés.

```
[1] Pattern                         ::= LocationPathPattern^T2
                                        | Pattern^T1 '|'
                                        LocationPathPattern^T2
[2] LocationPathPattern             ::= '/' RelativePathPattern^T4?
                                        | IdKeyPattern^T3 (('/' | '//')
                                        RelativePathPattern^T4)?
                                        | '//'? RelativePathPattern^T4
[3] IdKeyPattern                    ::= 'id' '(' Literal^P29 ')'
                                        | 'key' '(' Literal^P29 ','
                                        Literal^P29 ')'
[4] RelativePathPattern             ::= StepPattern^T5
                                        | RelativePathPattern^T4 '/'
                                        StepPattern^T5
                                        | RelativePathPattern^T4 '//'
                                        StepPattern^T5
[5] StepPattern                     ::= ChildOrAttributeAxisSpecifier^T6
                                        NodeTest^P7 Predicate^P8*
[6] ChildOrAttributeAxisSpecifier   ::= AbbreviatedAxisSpecifier^P13
                                        | ('child' | 'attribute')
                                        '::'
```

D.2. Éléments du vocabulaire XSLT par ordre alphabétique

Pour chaque élément nous donnons le titre et le numéro de la section où il est traité dans la recommandation XSLT, ainsi que sa syntaxe, indiquant l'endroit où l'élément peut être utilisé et ce qu'il peut contenir.

`xsl:apply-imports` Section 5.6 *Overriding Template Rules*.

```
<!-- catégorie : instruction -->
<xsl:apply-imports />
```

`xsl:apply-templates` Section 5.4 *Applying Template Rules*.

```
<!-- catégorie : instruction -->
<xsl:apply-templates
  select = node-set-expression
  mode = qname>
  <!-- contient : (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>
```

`xsl:attribute` Section 7.1.3 *Creating Attributes with `xsl:attribute`.*

```
<!-- catégorie : instruction -->
<xsl:attribute
  name = { qname }
  namespace = { uri-reference }>
  <!-- contient : template -->
</xsl:attribute>
```

`xsl:attribute-set` Section 7.1.4 *Named Attribute Sets.*

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:attribute-set
  name = qname
  use-attribute-sets = qnames>
  <!-- contient : xsl:attribute* -->
</xsl:attribute-set>
```

`xsl:call-template` Section 6 *Named Templates.*

```
<!-- catégorie : instruction -->
<xsl:call-template
  name = qname>
  <!-- contient : xsl:with-param* -->
</xsl:call-template>
```

`xsl:choose` Section 9.2 *Conditional Processing with `xsl:choose`.*

```
<!-- catégorie : instruction -->
<xsl:choose>
  <!-- contient : (xsl:when+, xsl:otherwise?) -->
</xsl:choose>
```

`xsl:comment` Section 7.4 *Creating Comments.*

```
<!-- catégorie : instruction -->
<xsl:comment>
  <!-- contient : template -->
</xsl:comment>
```

`xsl:copy` Section 7.5 *Copying.*

```
<!-- catégorie : instruction -->
<xsl:copy
  use-attribute-sets = qnames>
  <!-- contient : template -->
</xsl:copy>
```

`xsl:copy-of` Section 11.3 *Using Values of Variables and Parameters with `xsl:copy-of`.*

```
<!-- catégorie : instruction -->
<xsl:copy-of
  select = expression />
```

`xsl:decimal-format` Section 12.3 *Number Formatting*.

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:decimal-format
  name = qname
  decimal-separator = char
  grouping-separator = char
  infinity = string
  minus-sign = char
  NaN = string
  percent = char
  per-mille = char
  zero-digit = char
  digit = char
  pattern-separator = char />
```

`xsl:element` Section 7.1.2 *Creating Elements with* `xsl:element`.

```
<!-- catégorie : instruction -->
<xsl:element
  name = { qname }
  namespace = { uri-reference }
  use-attribute-sets = qnames>
  <!-- contient : template -->
</xsl:element>
```

`xsl:fallback` Section 15 *Fallback*.

```
<!-- catégorie : instruction -->
<xsl:fallback>
  <!-- contient : template -->
</xsl:fallback>
```

`xsl:for-each` Section 8. *Repetition*.

```
<!-- catégorie : instruction -->
<xsl:for-each
  select = node-set-expression>
  <!-- contient : (xsl:sort*, template) -->
</xsl:for-each>
```

`xsl:if` Section 9.1 *Conditional Processing with* `xsl:if`.

```
<!-- catégorie : instruction -->
<xsl:if
  test = boolean-expression>
  <!-- contient : template -->
</xsl:if>
```

`xsl:import` Section 2.6.2 *Stylesheet Import*.

```
<xsl:import
  href = uri-reference />
```

`xsl:include` Section 2.6.1 *Stylesheet Inclusion*.

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:include
  href = uri-reference />
```

`xsl:key` Section 12.2 *Keys*.

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:key
  name = qname
  match = pattern
  use = node-set-expression />
```

`xsl:message` Section 13 *Messages*.

```
<!-- catégorie : instruction -->
<xsl:message
  terminate = "yes" | "no">
  <!-- contient : template -->
</xsl:message>
```

`xsl:namespace-alias` Section 7.1.1 *Literal Result Elements*.

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:namespace-alias
  stylesheet-prefix = prefix | "#default"
  result-prefix = prefix | "#default" />
```

`xsl:number` Section 7.7 *Numbering*.

```
<!-- catégorie : instruction -->
<xsl:number
  level = "single" | "multiple" | "any"
  count = pattern
  from = pattern
  value = number-expression
  format = { string }
  lang = { nmtoken }
  letter-value = { "alphabetic" | "traditional" }
  grouping-separator = { char }
  grouping-size = { number } />
```

`xsl:otherwise` Section 9.2 *Conditional Processing with* `xsl:choose`.

```
<xsl:otherwise>
  <!-- contient : template -->
</xsl:otherwise>
```

`xsl:output` Section 16 *Output*.

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:output
```

```
  method = "xml" | "html" | "text" | qname-but-not-ncname
  version = nmtoken
  encoding = string
  omit-xml-declaration = "yes" | "no"
  standalone = "yes" | "no"
  doctype-public = string
  doctype-system = string
  cdata-section-elements = qnames
  indent = "yes" | "no"
  media-type = string />
```

`xsl:param` Section 11 *Variables and Parameters*.

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:param
  name = qname
  select = expression>
  <!-- contient : template -->
</xsl:param>
```

`xsl:preserve-space` Section 3.4 *Whitespace Stripping*.

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:preserve-space
  elements = tokens />
```

`xsl:processing-instruction` Section 7.3 *Creating Processing Instructions*.

```
<!-- catégorie : instruction -->
<xsl:processing-instruction
  name = { ncname }>
  <!-- contient : template -->
</xsl:processing-instruction>
```

`xsl:sort` Section 10 *Sorting*.

```
<xsl:sort
  select = string-expression
  lang = { nmtoken }
  data-type = { "text" | "number" }
  order = { "ascending" | "descending" }
  case-order = { "upper-first" | "lower-first" } />
```

`xsl:strip-space` Section 3.4 *Whitespace Stripping*.

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:strip-space
  elements = tokens />
```

`xsl:stylesheet` Section 2.2 *Stylesheet Element* (comme `xsl:transform`).

```
<xsl:stylesheet
  id = id
```

```
  extension-element-prefixes = tokens
  exclude-result-prefixes = tokens
  version = number>
  <!-- contient : (xsl:import*, élément de niveau supérieurs) -->
</xsl:stylesheet>
```

`xsl:template` Section 5.3 *Defining Template Rules.*

```
<!-- catégorie : élément de niveau supérieur -->
<xsl:template
  match = pattern
  name = qname
  priority = number
  mode = qname>
  <!-- contient : (xsl:param*, template) -->
</xsl:template>
```

`xsl:text` Section 7.2 *Creating Text.*

```
<!-- catégorie : instruction -->
<xsl:text
  disable-output-escaping = "yes" | "no">
  <!-- contient : #PCDATA -->
</xsl:text>
```

`xsl:transform` Section 2.2 *Stylesheet Element* (alias pour `xsl:stylesheet`).

```
<xsl:transform
  id = id
  extension-element-prefixes = tokens
  exclude-result-prefixes = tokens
  version = number>
  <!-- contient : (xsl:import*, élément de niveau supérieurs) -->
</xsl:transform>
```

`xsl:value-of` Section 7.6.1 *Generating Text with `xsl:value-of`.*

```
<!-- catégorie : instruction -->
<xsl:value-of
  select = string-expression
  disable-output-escaping = "yes" | "no" />
```

`xsl:variable` Section 11 *Variables and Parameters.*

```
<!-- catégorie : élément de niveau supérieur -->
<!-- catégorie : instruction -->
<xsl:variable
  name = qname
  select = expression>
  <!-- contient : template -->
</xsl:variable>
```

`xsl:when` Section 9.2 *Conditional Processing with* `xsl:choose`.

```
<xsl:when
  test = boolean-expression>
  <!-- contient : template -->
</xsl:when>
```

`xsl:with-param` Section 11.6 *Passing Parameters to Templates*.

```
<xsl:with-param
  name = qname
  select = expression>
  <!-- contient : template -->
</xsl:with-param>
```

E. Fonctions XPath et XSLT par ordre alphabétique

Pour chaque fonction, nous spécifions son type et la section où elle est traitée dans les recommandations XPath or XSLT.

`boolean(`*`object`*`)` booléenne (XPath, 4.3 *Boolean Functions*).

`ceiling(`*`number`*`)` numérique (XPath, 4.4 *Number Functions*).

`concat(`*`string`*`,`*`string`*`,`*`string`*`*)` chaîne de caractères (XPath, 4.2 *String Functions*).

`contains(`*`string`*`,`*`string`*`)` chaîne de caractères (XPath, 4.2 *String Functions*).

`count(`*`node-set`*`)` ensemble de nœuds (XPath, 4.1 *Node Set Functions*).

`current()` ensemble de nœuds (XSLT, 12.4 *Miscellaneous Additional Functions*).

`document(`*`object`*`,` *`node-set`*`?)` ensemble de nœuds (XSLT, 12.1 *Multiple Source Documents*).

`element-available(`*`string`*`)` booléenne (XSLT, 15 *Fallback*).

`false()` booléenne (XPath, 4.3 *Boolean Functions*).

`floor(`*`number`*`)` numérique (XPath, 4.4 *Number Functions*).

`format-number(`*`number`*`,`*`string`*`,`*`string`*`?)`chaîne de caractères (XSLT, 12.3 *Number Formatting*).

`function-available(`*`string`*`)` booléenne (XSLT, 15 *Fallback*).

`generate-id(`*`node-set`*`?)` ensemble de nœuds (XSLT, 12.4 *Miscellaneous Additional Functions*).

`id(`*`object`*`)` ensemble de nœuds (XPath, 4.1 *Node Set Functions*).

`key(`*`string`*`,`*`object`*`)` ensemble de nœuds (XSLT, 12.2 *Keys*).

`lang(`*`string`*`)` booléenne (XPath, 4.3 *Boolean Functions*).

`last()` ensemble de nœuds (XPath, 4.1 *Node Set Functions*).

`local-name(`*`node-set?`*`)` ensemble de nœuds (XPath, 4.1 *Node Set Functions*).
`name(`*`node-set?`*`)` ensemble de nœuds (XPath, 4.1 *Node Set Functions*).
`namespace-uri(`*`node-set?`*`)` ensemble de nœuds (XPath, 4.1 *Node Set Functions*).
`normalize-space(`*`string?`*`)` chaîne de caractères (XPath, 4.2 *String Functions*).
`not(`*`boolean`*`)` booléenne (XPath, 4.3 *Boolean Functions*).
`number(`*`object?`*`)` numérique (XPath, 4.4 *Number Functions*).
`position()` ensemble de nœuds (XPath, 4.1 *Node Set Functions*).
`round(`*`number`*`)` numérique (XPath, 4.4 *Number Functions*).
`starts-with(`*`string`*`,`*`string`*`)` chaîne de caractères (XPath, 4.2 *String Functions*).
`string(`*`object?`*`)` chaîne de caractères (XPath, 4.2 *String Functions*).
`string-length(`*`string?`*`)` numérique (XPath, 4.2 *String Functions*).
`substring(`*`string`*`,`*`number`*`,`*`number?`*`)` chaîne de caractères (XPath, 4.2 *String Functions*).
`substring-after(`*`string`*`,`*`string`*`)` chaîne de caractères (XPath, 4.2 *String Functions*).
`substring-before(`*`string`*`,`*`string`*`)` chaîne de caractères (XPath, 4.2 *String Functions*).
`sum(`*`node-set`*`)` numérique (XPath, 4.4 *Number Functions*).
`system-property(`*`string`*`)` object système identifié comme un nom qualifié (XSLT, 12.4 *Miscellaneous Additional Functions*).
`translate(`*`string`*`,`*`string`*`,`*`string`*`)` chaîne de caractères (XPath, 4.2 *String Functions*).
`true()` booléenne (XPath, 4.3 *Boolean Functions*).
`unparsed-entity-uri(`*`string`*`)` chaîne de caractères (XSLT, 12.4 *Miscellaneous Additional Functions*).

F. Terminologie bilingue XML

Il n'est pas toujours facile de traduire des termes techniques des spécifications W3C. En particulier, pour les textes de XML, XSL, XLink et XPointer, nous avons quelquefois dû créer un néologisme. Heureusement, nous étions aidés par le fait que la norme ISO SGML est une des rares à être traduite en français[38]. En plus les volontaires qui ont contribué à la traduction de la spécification XML (voir page 191) ont augmenté considérablement la liste[5]. Le tableau qui suit contient des termes puisés dans ces deux listes ainsi que quelques nouvelles traductions de termes que nous avons utilisées

5. Cette liste est en grande partie l'œuvre de Patrick ANDRIES et François YERGEAU et se trouve à l'URL `http://babel.alis.com/web_ml/xml/termino.html`.

dans le présent texte. Il est clair que ces dernières n'ont qu'une valeur subjective et que nous adopterons volontiers dans le futur une traduction « officielle » dès qu'elle existera.

TERME ANGLAIS	TRADUCTION PROPOSÉE
ampersand	esperluette
attribute	attribut
attribute-list declaration	déclaration de liste d'attributs
attribute specification	spécification d'attribut, stipulation d'attribut
attribute-value normalization	normalisation de valeur d'attribut
base character	caractère de base
big-endian	grand-boutien
bypassed	outrepassé
CDATA section	section CDATA
character	caractère
character data	données textuelles
character encoding	codage des caractères
character reference	appel de caractère
child element	sous-élément
choice list of content particles	liste d'options de particules (de contenu)
combining character	caractère jonctif
compatibility area	zone de compatibilité
compatibility decomposition	décomposition de compatibilité
compatibility formatting tag	balise de formatage de compatibilité
conditional section	section conditionnelle
consexte node	nœud contexte
constraint	contrainte
construct	construction, production
content particle	particule de contenu, particule
cookie	témoin de connexion, mouchard
declaration	déclaration
document	document
document element	élément document
document type declaration	déclaration de type de document
document type definition	définition de type de document
element	élément
element content	contenu élémentaire
element-content model	modèle de contenu élémentaire
empty-element tag	balise d'élément vide
end-tag	balise de fin, balise fermante
entity	entité
entity reference	appel d'entité
enumerated type	type énuméré

TERME ANGLAIS	TRADUCTION PROPOSÉE
escape (to)	masquer
extender	modificateur de lettre
external entity	entité externe
follow set	ensemble des suivants
generic identifier	identificateur générique
internal entity	entité interne
litteral entity value	valeur littérale d'entité
little-endian	petit-boutien
location path	chemin de localisation
locator	localisateur, élément de localisation
markup	balisage
markup declaration	déclaration de balisage
match	correspondre, correspondance
match pattern	motif de correspondance
mixed content	contenu mixte
name	nom
name characters	caractères constitutifs de nom
name start characters	caractères initiaux de nom
name token	unité lexicale nominale, atome nominal
named template	modèle nommé
non-terminal	non-terminal
notation	notation
notation type	type notation
notify	signalé
numeric character reference	appel de caractère numérique
parameter entity	entité paramètre
parsed data	données parsées, données analysables
parsed entity	entité parsée, entité analysable
parsed textual data	données textuelles parsées, données textuelles analysées
pattern	motif
processing instruction	instruction de traitement
prolog	prologue
proper nesting	imbrication stricte
proxy	mandataire
pull	recherche ou extraction individuelle
push	distribution sélective
replacement text	texte de remplacement
root element	élément racine
rule, production, construct	production, règle
selection pattern	motif de sélection
sequence list of content particles	liste de suite de particules [de contenu]
standalone document	document autonome

TERME ANGLAIS	TRADUCTION PROPOSÉE
standardization	standardisation
start-tag	balise de début, balise ouvrante
set of attributes	jeu d'attributs
set of tokenized types	série de types atomiques
string expressions	expressions contenant des chaînes de caractères
string type	type chaîne [de caractères]
surrogates	substituts
system identifier	identificateur système
text declaration	déclaration de texte
token, tokenized, tokenizer	atome, atomisé, atomiseur
tokenized type	type atomique
trailing blanks	blancs de queue
unparsed entity	entité non-parsée, entité non-analysable
URL	adresse universelle
valid, validity, validation	valide, validité, validation
validating [XML] processor	processeur [XML] validateur
well-formed	bien formé
well-formedness constraint	contrainte de forme
white space	séparateur, du blanc
XML processor	processeur XML

G. Traduire un fichier HTML en LaTeX

Cette section présente une feuille de style XSL pour transformer un fichier HTML relativement simple en LaTeX. Habituellement, il faut traiter le fichier d'abord avec l'utilitaire `tidy`[28] pour générer du XHTML (une version XML de HTML). Cette feuille de style traite uniquement les éléments de HTML qui ne sont pas trop orientés vers une présentation visuelle. Pour ceux-là, le plus souvent, il faut compléter ce qui suit par du code *ad hoc* pour traiter les structures plus complexes (tables, formes, effets coloriés, polices spéciales, etc.).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY lt     "&#60;">
<!ENTITY gt     ">">
<!ENTITY amp    "&#38;">
<!ENTITY nbsp   " ">
]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.tug.org/latex">

<xsl:strip-space elements="a par"/>

<xsl:output method="text" indent="no" encoding="ISO-8859-1"/>
```

```
14
15  <xsl:template name="label">
16    <xsl:apply-templates/>
17    <xsl:text>}</xsl:text><!-- fermer } de la commande ...section{ -->
18    <xsl:if test="a[@name]">
19      <xsl:text>\label{</xsl:text>
20      <xsl:value-of select="a/@name"/>
21      <xsl:text>}&#xA;</xsl:text>
22    </xsl:if>
23  </xsl:template>
24
25  <!-- ========= Root node ============ -->
26  <xsl:template match="/">
27  <xsl:text>\documentclass{article}
28  \usepackage{fromhtml}
29  \usepackage[T1]{fontenc}
30  \begin{document}
31  </xsl:text>
32  <xsl:apply-templates/>
33  <xsl:text>\end{document}&#xA;</xsl:text>
34  </xsl:template>
35
36  <!-- ========= Section headings ============ -->
37  <xsl:template match="h1">
38    <xsl:text>&#xA;\section*{</xsl:text>
39    <xsl:call-template name="label"/>
40  </xsl:template>
41  <xsl:template match="h2">
42    <xsl:text>&#xA;\subsection*{</xsl:text>
43    <xsl:call-template name="label"/>
44  </xsl:template>
45  <xsl:template match="h3">
46    <xsl:text>&#xA;\subsubsection*{</xsl:text>
47    <xsl:call-template name="label"/>
48  </xsl:template>
49  <xsl:template match="h4">
50    <xsl:text>&#xA;\paragraph{</xsl:text>
51    <xsl:call-template name="label"/>
52  </xsl:template>
53  <xsl:template match="h5|h6">
54    <xsl:text>&#xA;\subparagraph{</xsl:text>
55    <xsl:call-template name="label"/>
56  </xsl:template>
57
58  <!-- ========= Emphasis ============ -->
59
60  <xsl:template match="font"><!-- just continue -->
61    <xsl:apply-templates/>
62  </xsl:template>
63  <xsl:template match="font/@color">
64    <xsl:text>{\bfseries\itshape </xsl:text>
65    <xsl:apply-templates/>
66    <xsl:text>}</xsl:text>
67  </xsl:template>
68  <xsl:template match="font"><!-- just continue -->
69    <xsl:apply-templates/>
70  </xsl:template>
71  <xsl:template match="cite|em|i|var">
72    <xsl:text>\emph{</xsl:text>
73    <xsl:apply-templates/>
74    <xsl:text>}</xsl:text>
75  </xsl:template>
76  <xsl:template match="b|strong">
```

```
77    <xsl:text>\textbf{</xsl:text>
78    <xsl:apply-templates/>
79    <xsl:text>}</xsl:text>
80  </xsl:template>
81  <xsl:template match="tt|code">
82    <xsl:text>\texttt{</xsl:text>
83    <xsl:apply-templates/>
84    <xsl:text>}</xsl:text>
85  </xsl:template>
86  <xsl:template match="big">
87    <xsl:text>{\large </xsl:text>
88    <xsl:apply-templates/>
89    <xsl:text>}</xsl:text>
90  </xsl:template>
91  <xsl:template match="small">
92    <xsl:text>{\small </xsl:text>
93    <xsl:apply-templates/>
94    <xsl:text>}</xsl:text>
95  </xsl:template>
96  <xsl:template match="center">
97    <xsl:text>&#xA;\begin{center}&#xA;</xsl:text>
98    <xsl:apply-templates/>
99    <xsl:text>&#xA;\end{center}</xsl:text>
100 </xsl:template>
101 <!-- ======== Cross-references ======== -->
102 <xsl:template match="ref">
103   <xsl:text>\ref{</xsl:text>
104   <xsl:value-of select="@refid"/>
105   <xsl:text>}</xsl:text>
106 </xsl:template>
107 <!-- ====== quotes, footnotes, verbatim ====== -->
108 <xsl:template match="blockquote">
109   <xsl:text>&#xA;\begin{quote}</xsl:text>
110   <xsl:apply-templates/>
111   <xsl:text>\end{quote}&#xA;</xsl:text>
112 </xsl:template>
113 <xsl:template match="pre">
114   <xsl:text>&#xA;\begin{verbatim}</xsl:text>
115   <xsl:apply-templates/>
116   <xsl:text>\end{verbatim}&#xA;</xsl:text>
117 </xsl:template>
118 <!-- ============ Lists ============== -->
119 <xsl:template match="dl">
120   <xsl:text>&#xA;\begin{description}&#xA;</xsl:text>
121   <xsl:apply-templates/>
122   <xsl:text>&#xA;\end{description}&#xA;</xsl:text>
123 </xsl:template>
124 <xsl:template match="dl/dt">
125   <xsl:text>&#xA;\item[</xsl:text>
126   <xsl:apply-templates/>
127   <xsl:text>]</xsl:text>
128 </xsl:template>
129 <xsl:template match="dl/dd">
130   <xsl:apply-templates/>
131   <xsl:if test="name(following::*[1]) = 'dd'">
132    <xsl:text>&#xA;\par&#xA;</xsl:text>
133   </xsl:if>
134 </xsl:template>
135   <xsl:template match="ol">
136   <xsl:text>&#xA;\begin{enumerate}&#xA;</xsl:text>
137   <xsl:apply-templates/>
138   <xsl:text>&#xA;\end{enumerate}&#xA;</xsl:text>
139 </xsl:template>
```

```
<xsl:template match="ul">
  <xsl:text>&#xA;\begin{itemize}&#xA;</xsl:text>
  <xsl:apply-templates/>
  <xsl:text>&#xA;\end{itemize}&#xA;</xsl:text>
</xsl:template>
<xsl:template match="ol/li">
  <xsl:text>&#xA;\item </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="ul/li">
  <xsl:text>&#xA;\item </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
<!-- ===== sub, sup -->
<xsl:template match="sub">
  <xsl:text>\(\sb{</xsl:text>
  <xsl:apply-templates/>
  <xsl:text>}\)</xsl:text>
</xsl:template>
<xsl:template match="sup">
  <xsl:text>\(\sp{</xsl:text>
  <xsl:apply-templates/>
  <xsl:text>}\)</xsl:text>
</xsl:template>
<!-- ===== a, br, code, hr, paragraph and url -->
<xsl:template match="a[@name]">
  <!-- traité par les headings <hi> -->
<xsl:apply-templates/>
</xsl:template>
  <xsl:template match="a[@href]">
  <xsl:apply-templates/>
  <xsl:choose>
  <xsl:when test="substring(@href,1,1)='#'">
    <xsl:text> (page \pageref{</xsl:text>
    <xsl:value-of select="@href"/>
    <xsl:text>})</xsl:text>
  </xsl:when>
  <xsl:otherwise>
    <xsl:text> (\url{</xsl:text>
    <xsl:value-of select="@href"/>
    <xsl:text>})</xsl:text>
  </xsl:otherwise>
</xsl:choose>
</xsl:template>
<xsl:template match="br">
  <xsl:text>\newline </xsl:text>
</xsl:template>
<xsl:template match="code|kbd|samp">
  <xsl:text>{\ttfamily </xsl:text>
  <xsl:apply-templates/>
  <xsl:text>}</xsl:text>
</xsl:template>
<xsl:template match="hr">
  <!-- ne rien faire -->
</xsl:template>
<xsl:template match="p">
  <xsl:text>&#xA;&#xA;</xsl:text>  <!-- ligne blanche -->
  <xsl:apply-templates/>
</xsl:template>
<!-- ===== tabular ===== -->
<xsl:template match="table">
<xsl:text>\begin{tabulary}{\textwidth}{</xsl:text>
  <xsl:for-each select="tr[1]/td">
```

```
203       <xsl:choose>
204       <xsl:when test="@align='right'">
205         <xsl:text>R</xsl:text>
206       </xsl:when>
207       <xsl:when test="@align='center'">
208         <xsl:text>C</xsl:text>
209       </xsl:when>
210       <xsl:otherwise>
211         <xsl:text>L</xsl:text>
212       </xsl:otherwise>
213       </xsl:choose>
214     </xsl:for-each>
215     <xsl:text>L}</xsl:text>
216     <xsl:apply-templates/>
217   <xsl:text>\end{tabulary}</xsl:text>
218   </xsl:template>
219
220   <xsl:template match="tr|th">
221     <xsl:apply-templates/>
222     <xsl:text>\\\empty&#xA;</xsl:text>
223   </xsl:template>
224
225   <xsl:template match="tr/td|tr/th">
226     <xsl:apply-templates/>
227     <xsl:text>\TaB </xsl:text>
228   </xsl:template>
229
230   <xsl:template match="tr/td[position()=last()]" priority="2">
231     <xsl:apply-templates/>
232   </xsl:template>
233
234   <xsl:template match="tr/th[position()=last()]" priority="2">
235     <xsl:apply-templates/>
236   </xsl:template>
237   <!-- ======= "floats" and their contents ========= -->
238   <xsl:template match="img">
239     <xsl:text>&#xA;\includegraphics{</xsl:text>
240     <xsl:value-of select="@src"/>
241     <xsl:text>}&#xA;</xsl:text>
242   </xsl:template>
243
244   <!-- région attrape tout -->
245
246   <xsl:template match="form|head">
247    <!-- éliminer les forms et le head -->
248   </xsl:template>
249
250   <xsl:template match="html|body|span">
251     <xsl:apply-templates/><!-- continuer -->
252   </xsl:template>
253
254   <xsl:template match="text()">
255     <xsl:value-of select="translate(.,' ğ',' ')"/>
256   </xsl:template>
257
258   <xsl:template match="*">
259     <xsl:text>&#xA;\par ******* Élément non reconnu ******* </xsl:text>
260     <xsl:value-of select="name(.)"/>
261     <xsl:text> ****&#xA;</xsl:text>
262     <xsl:apply-templates/>
263   \end{flushleft}
264   </xsl:template>
265   </xsl:stylesheet>
```

Remerciements

Je tiens à remercier Sebastian RAHTZ pour une collaboration fructueuse, des discussions stimulantes et pour l'aide technique avec quelques figures récalcitrantes. David CARLISLE était toujours prêt à répondre à mes questions XSL, même les plus stupides.

Plusieurs personnes ont eu la gentillesse de relire une version préliminaire de cet article : Lisiane BESSON, Bernard GAULLE, Michèle JOUHET, Maurice LAUGIER.

Par ses commentaires enrichissants et ses suggestions constructives Jacques ANDRÉ a considérablement contribué à améliorer le texte du présent article.

Évidemment j'assume, comme auteur, la responsabilité de toutes les fautes et imprécisions restantes.

Bibliographie

[1] Alis Technologies inc. *Liste des noms des caractères ISO 10646-1 (1993).* `http://babel.alis.com/codage/iso10646/index.html`

[2] Jacques ANDRÉ et Michel GOOSSENS. « Codage des caractères et multi-linguisme : de l'Ascii à Unicode et ISO/IEC-10646. » *Cahiers GUTenberg*, n° 20, pages 1–54, mai 1995.

[3] Ronald BOURRET. XML *and Databases* `http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLAndDatabases.htm`

[4] Lou BURNARD et C.M. SPERBERG-MCQUEEN. « La TEI simplifiée : une introduction au codage des textes électroniques en vue de leur échange. » *Cahiers GUTenberg*, n° 24, pages 23–152, juin 1996.

[5] James CLARK. *`xt`, an implementation in Java of XSL Transformations.* `http://www.jclark.com/xml/xt.html`

[6] James CLARK. *`nsgmls`, an SGML System Conforming to ISO Standard 8879.* `http://www.jclark.com/sp/nsgmls.htm`

[7] James CLARK. *`nsgmls`, Output Format.* `http://jclark.com/sp/sgmlsout.htm`

[8] Robin COVER (Oasis). *The SGML/XML Web Page (XSL).* `http://www.oasis-open.org/cover/xsl.html`

[9] Dale DOUGHERTY. *The Making of the DocBook DTD.* `http://www.xml.com/pub/1999/10/docbook/docbook-making.html`

[10] Daniel GLAZMAN. *CSS2 Feuilles de styles HTML.* Éditions Eyrolles, Paris, 1999.

[11] Charles F. GOLDFARB. *The SGML Handbook.* Oxford University Press, 1992.

[12] Michel GOOSSENS. « Introduction pratique à SGML. » *Cahiers GUTenberg*, n° 19, pages 27–58, janvier 1995.

[13] Michel GOOSSENS, Frank MITTELBACH et Alexander SAMARIN. *The LaTeX Companion*. Addison-Wesley, Reading, 1995.

[14] Michel GOOSSENS, Sebastian RAHTZ et Frank MITTELBACH. *The LaTeX Graphics Companion*. Addison-Wesley, Reading, 1997.

[15] Michel GOOSSENS et Sebastian RAHTZ. *The LaTeX Web Companion*. Addison-Wesley, Reading, 1999.

[16] Eric van HERWIJNEN. *SGML* PRATIQUE. International Thomson Publishing France, Paris, 1995.

[17] Ken HOLMAN. *Practical Transformation Using XSLT and XPath.* `http://www.CraneSoftwrights.com/training/index.htm`

[18] IBM alphaWorks. *LotusXSL, an XSL processor in Java.* `http://www.alphaworks.ibm.com/tech/lotusxsl`

[19] IBM alphaWorks. *A prototype Scalable Vector Graphics (SVG) viewer.* `http://www.alphaworks.ibm.com/tech/svgview`

[20] IBM alphaWorks. *`xmlviewer`, a Java class for viewing XML documents.* `http://www.alphaworks.ibm.com/tech/xmlviewer`

[21] IBM alphaWorks. *XML4J, XML parser for Java.* `http://www.alphaworks.ibm.com/tech/xml4j`

[22] The Internet Society. Tim BERNERS-LEE *et al. Uniform Resource Identifiers (URI): Generic Syntax.* `http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2396.txt`

[23] Leslie Michel LAMPORT. *LaTeX User's Guide and Reference Manual, Second Edition*. Addison-Wesley, Reading, 1994.

[24] Alain MICHARD. *XML Langage et applications*. Éditions Eyrolles, Paris, 1999.

[25] Nic MILOSLAV. *Introduction to XSL.* `http://zvon.vscht.cz/HTMLonly/XSLTutorial/Books/Book1/bookInOne.html`

[26] Mulberry Technologies. *XSL-List — Open Forum on XSL.* `http://www.mulberrytech.com/xsl/xsl-list/`

[27] Dave PAWSON. *XSL Frequently Asked Questions.* `http://freespace.virgin.net/ben.pawson/`

[28] David RAGGETT. *Clean up your Web pages with HTML Tidy.* `http://www.w3.org/People/Raggett/tidy/`

[29] Sebastian RAHTZ. *Passive TeX.* `http://users.ox.ac.uk/~rahtz/passivetex/`

[30] Sebastian RAHZT et Michel GOOSSENS. *PassiveTeX: XML and TeX, doing it together . . .* `http://wwwinfo.cern.ch/asdoc/WWW/publications/xmldev99/passivetex.html`

[31] Sebastian RAHTZ. *TEI and XSL.*
`http://users.ox.ac.uk/~rahtz/tei/`

[32] Gaspar SINAI et Roman CZYBORRA. *Yudit Unicode editor.*
`http://czyborra.com/yudit/`

[33] C.M. SPERBERG-MCQUEEN et Lou BURNARD (rédacteurs).
Guidelines for Electronic Text Encoding and Interchange.
Chicago, Oxford: Text Encoding Initiative, 1994.

[34] James TAUBER. *FOP: A Formatting Object to PDF Formatter/Renderer.*
`http://www.jtauber.com/fop/`

[35] THE UNICODE CONSORTIUM. *The Unicode Standard, Version 2.0.*
Addison-Wesley, Reading, 1996.

[36] Norman WALSH (Oasis) *The Docbook DTD and stylesheets.*
`http://nwalsh.com/docbook/index.html`

[37] International Organization for Standardization. *Hypermedia/Time-based Structuring Language (Hytime).* ISO 10744, ISO Geneva, 1992.

[38] Organisation internationale de normalisation. *Langage normalisé de balise généralisé (SGML).* ISO 8879-1986(F), ISO Genève, 1986.

[39] World Wide Web Consortium, Håkon Wium LIE et Bert BOS (rédacteurs).
Cascading Style Sheets, level 1.
`http://www.w3.org/TR/REC-CSS1`

[40] World Wide Web Consortium, Håkon Wium LIE, Bert BOS, Chris LILLEY et Ian JACOBS (rédacteurs). *Cascading Style Sheets, level 2.*
`http://www.w3.org/TR/REC-CSS2`

[41] World Wide Web Consortium. Tim BRAY, Dave HOLLANDER et Andrew LAYMAN (rédacteurs). *Namespaces in XML.*
`http://www.w3.org/TR/REC-xml-names`

[42] World Wide Web Consortium. *QL'98 - The Query Languages Workshop.*
`http://www.w3.org/TandS/QL/QL98/Overview.html`

[43] World Wide Web Consortium. Jon FERRAIOLO (rédacteur). *Scalable Vector Graphics (SVG) 1.0 Specification. Working Draft.*
`http://www.w3.org/TR/SVG/`

[44] World Wide Web Consortium. Murray ALTHEIM et Shane MCCARRON (rédacteurs). *XHTML 1.1 - Module-based XHTML.*
`http://www.w3.org/TR/xhtml11/`

[45] World Wide Web Consortium. Murray ALTHEIM, Frank BOUMPHREY, Sam DOOLEY, Shane MCCARRON et Ted WUGOFSKI (rédacteurs).
Modularization of XHTML.
`http://www.w3.org/TR/xhtml-modularization/`

[46] World Wide Web Consortium. Tim BRAY, Jean PAOLI et C. M. SPERBERG-MCQUEEN (rédacteurs). *Extensible Markup Language (XML) 1.0.* `http://www.w3.org/TR/REC-xml`

[47] World Wide Web Consortium. Steve DEROSE, David ORCHARD et Ben TRAFFORD (rédacteurs). *XML Linking Language (XLink). Working Draft.* `http://www.w3.org/TR/xlink`

[48] World Wide Web Consortium, James CLARK et Steve DEROSE (rédacteurs). *XML Path Language (XPath), Version 1.0 (W3C Proposed recommendation).* `http://www.w3.org/TR/xpath`

[49] World Wide Web Consortium. Steve DEROSE et Ron DANIEL Jr. (rédacteurs) *XML Pointer Language (XPointer). Working Draft.* `http://www.w3.org/TR/WD-xptr`

[50] World Wide Web Consortium. Henry S. THOMPSON, David BEECH, Murray MALONEY et Noah MENDELSOHN (rédacteurs). *XML Schema Part 1: Structures.* `http://www.w3.org/TR/xmlschema-1`

[51] World Wide Web Consortium. Paul V. BIRON et Ashok MALHOTRA (rédacteurs). *XML Schema Part 2: Datatypes.* `http://www.w3.org/TR/xmlschema-2`

[52] World Wide Web Consortium, Stephen DEACH (rédacteur). *Extensible Stylesheet Language (XSL), Version 1.0 (W3C Working Draft).* `http://www.w3.org/TR/WD-xsl`

[53] World Wide Web Consortium, James CLARK (rédacteur). *XSL Transformations (XSLT), Version 1.0 (W3C Proposed recommendation).* `http://www.w3.org/TR/xslt`