Cahiers GUTenberg

DE XML À PDF VIA XMLTEX, XSLT ET PASSIVETEX

David Carlisle, Michel Goossens, Sebastian Rahtz

Cahiers GUTenberg, nº 35-36 (2000), p. 79-114.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_2000___35-36_79_0>

De XML à PDF via `xmltex`, XSLT et PassiveTEX

David Carlisle [1], Michel Goossens [2] etSebastian Rahtz [3]

[1] Numerical Algorithms Group, Oxford, OX2 8DR, Royaume-uni
davidc@nag.co.uk
[2] CERN, division IT, CH-1211 Genève 23, Suisse
michel.goossens@cern.ch
[3] Oxford University Computing Services, Oxford, OX2 6NN, Royaume-uni
sebastian.rahtz@oucs.ox.ac.uk

Mot-clés : DocBook, PassiveTEX, TEI, `xmltex`, XML, XSL, XSL-FO, XSLT.

Résumé.

Cet article introduit `xmltex`, un ensemble de macros TEX qui analyse un document XML et le compose piloté par des fichiers de configuration. Nous parlons également de PassiveTEX, une bibliothèque de macros TEX basée sur `xmltex` qui prend un document XML contenant des objets de formatage XSL et génère une sortie PDF ou DVI. Nous comparons ces deux approches avec une traduction directe du fichier source XML en LATEX. Nous montrons des exemples de ces techniques pour les DTD TEI, DocBook et MathML. L'annexe décrit plus en détail les commandes `xmltex`

Abstract.

This article introduces `xmltex`, a TEX macro package that parses an XML document and typesets it under the control of configuration files. We also discuss PassiveTEX, a library of TEX macros based on `xmltex`, that processes XML documents containing XSL formatting objects and generates PDF or DVI output. We compare these two approaches with a direct translation of the XML source file into LATEX. We show examples of these techniques for the TEI, DocBook and MathML DTDs. The appendix gives details about the `xmltex` commands.

1. Introduction

Actuellement, de plus en plus d'information circule sur l'Internet, en grande partie sous forme de HTML visualisé avec un butineur. Souvent, quand le lecteur veut imprimer l'information sur l'écran il est confronté à plusieurs problèmes, dont la qualité de la mise en page et la sérialisation de l'arbre HTML

sur un support linéaire comme le papier sont parmi les plus importants. Dans cet article nous proposons plusieurs approches basées sur XML et TEX pour résoudre ces problèmes.

Le choix de TEX comme moteur de composition se justifie par le fait que TEX traite les mathématiques de façon naturelle, permet la gestion de plusieurs langues (césures, règles typographiques, codages et alphabets mixtes, composition droite-gauche, etc.), propose un modèle structurel complexe et assez complet pour l'information tabulaire et parvient sans problèmes à traiter les documents très longs (comme la facturation des centaines de milliers d'abonnés au téléphone).

La première partie de l'article décrit un recueil de poèmes de Verlaine (voir [7], section 6.3) balisé en XML d'après la DTD TEI (*Text Encoding Initiative* [3]). Trois approches, toutes basées sur TEX, pour composer ce document sont étudiées.

Dans la deuxième partie, nous nous intéressons à un document XML ayant un contenu plus technique et quelques formules mathématiques. Le document a été balisé en utilisant les DTD *DocBook* [18] et MathML [20].

En nous inspirant de ces deux cas, nous concluerons que XML est un format idéal pour gérer, échanger et sauvegarder les documents électroniques en permettant une utilisation optimale de l'information.

Pour comprendre en détail le code TEX des exemples il est utile de se référer à l'annexe, qui contient une description des commandes `xmltex`.

Cet article a été préparé complètement en XML avec un balisage défini par la DTD TEI. Ce document source a ensuite été composé avec TEX en suivant la procédure décrite à la section 2 et en ajoutant la classe LATEX `cah-gut.cls` pour garantir une mise en page dans le style des *Cahiers GUtenberg*.

2. Composer un fichier XML directement avex `xmltex`

`xmltex` [4] est un programme d'analyse non-validant pour les documents XML composés suivant la norme des domaines nominaux (*namespaces*) [21]. Le système est composé d'une séries de macros TEX. `xmltex` peut se limiter à une analyse simple du fichier source XML : développement des appels d'entités, normalisation des déclarations de domaines nominaux, etc. Dans ce cas, seule une trace du déroulement de l'analyse sur le terminal sera générée. Par contre, plus généralement, `xmltex` peut traiter le résultat de l'analyse pour générer du code source TEX afin de composer le document. La distribution `xmltex` propose une série de commandes (voir l'annexe) pour associer du code source

TEX avec le début et la fin des éléments XML, les attributs, les instructions de traitement et avec les données textuelles Unicode, le codage de base utilisé par XML.

Il est donc important de réaliser que `xmltex` se limite à l'interpétation du document dans le contexte XML. C'est l'utilisateur qui à l'aide des commandes de traitement proposées devra définir comment le document XML doit être composé, à l'aide de LATEX ou tout autre format TEX.

2.1. Utiliser `xmltex` avec LATEX

Dans ce qui suit nous utiliserons le fichier `verlainetei.xml` (introduit à la section 6.3 de [7]) pour étudier le fonctionnement du processeur `xmltex`. Pour plus de clarté nous reproduisons ce fichier ci-dessous :

```
 1: <?xml version="1.0" encoding="ISO-8859-1"?>
 2: <!DOCTYPE TEI.2 SYSTEM "teixlite.dtd" [
 3: <!ENTITY dash "&#x2010;">
 4: <!ENTITY mdash "&#x2014;">
 5: <!ENTITY oelig "&#x0153;">
 6: <!ENTITY Saturniens SYSTEM "verlaineps.xml">
 7: <!ENTITY Galantes    SYSTEM "verlainefg.xml">
 8: ]>
 9: <TEI.2>
10: <teiHeader type="text" status="new">
11: <fileDesc>
12: <titleStmt>
13: <title>Divers recueils de poèmes de Paul Verlaine</title>
14: </titleStmt>
15: <publicationStmt>
16: <distributor>Exemple TEI préparé par mg pour GUTenberg</distributor>
17: </publicationStmt>
18: <sourceDesc default="NO">
19: <p>http://www.ambafrance.org/FLORILEGE/verlaine/a.html</p>
20: </sourceDesc>
21: </fileDesc>
22: </teiHeader>
23: <text>
24: <front>
25: <titlePage>
26: <docTitle>
27: <titlePart>Recueil de poèmes de Paul Verlaine</titlePart>
28: </docTitle>
29: <docAuthor>Michel Goossens</docAuthor>
30: <docDate>Octobre 1999</docDate>
31: </titlePage>
32: </front>
33: <body>
34: &Saturniens;
35: &Galantes;
```

```
36: </body>
37: </text>
38: </TEI.2>
```

LaTeX ne reconnaît pas un document en format XML, il ne connaît que les commandes TeX. Ainsi nous devons d'abord *préparer* LaTeX à accepter une syntaxe XML, ce que nous faisons en créant un fichier `verlainetei.tex` qui contient :

```
1: \def\xmlfile{verlainetei.xml}
2:   \input xmltex.tex
3:   \end{document}
```

La ligne 1 définit le nom du fichier XML à traiter, puis la ligne 2 passe le contrôle à `xmltex`. Il est important de ne mettre aucune autre commande dans ce fichier !

Après l'installation des fichiers de la distribution `xmltex` [4] sur votre système informatique à un endroit accessible par TeX, il suffit d'exécuter LaTeX à l'aide de la commande adéquate dans votre environnement TeX, par exemple :

latex verlainetei

2.2. Exemple de traitement `xmltex`

Comme évoqué ci-dessus `xmltex` se limite à l'analyse d'un document source XML, il ne le compose pas. C'est l'utilisateur de `xmltex` qui doit indiquer le traitement désiré pour chaque élément XML.

Au début du chargement de `xmltex`, TeX essaie de trouver un fichier de configuration avec le même nom que le fichier source XML mais ayant une extension `.cfg`, c.-à-d. dans notre cas il cherchera un fichier `verlainetei.cfg`. Celui-ci contient :

```
1: \NAME{TEI.2} {verlaine.xmt}
2: \UnicodeCharacter{x2010}{-}
3: \UnicodeCharacter{x2014}{-}
4: \UnicodeCharacter{x0153}{{\oe}}
```

La ligne 1 indique que lorsque `xmltex` rencontrera la balise de début `TEI.2`, correspondant à l'élément racine du document `verlainetei.xml`, il chargera le fichier `verlaine.xmt`. Les lignes 2 à 4 donnent la traduction en langage TeX des caractères Unicode : tiret de taille moyenne, tiret de taille longue, ligature `oe` (voir les définitions des entités `dash`, `mdash` et `oelig` aux lignes 3 à 5 du fichier `verlainetei.xml` donné en section 2.1).

En lisant le fichier source XML `verlainetei.xml` à la ligne 9 `xmltex` rencontre la balise de début `<TEI.2>` puis charge le fichier `verlaine.xmt`, qui contient les lignes suivantes :

```
 1: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 2: %% verlaine.xmt    (basé sur tei.xmt)       %%
 3: %% Michel Goossens (à utiliser avec xmltex) %%
 4: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 5:
 6: \XMLelement{TEI.2}{}
 7:   { \documentclass{article}
 8:     \usepackage{ifthen}
 9:     \usepackage{a4wide}
10:      \usepackage[T1]{fontenc}
11:     \begin{document}}
12:   {\end{document}}
13:
14: \XMLelement{teiHeader}{}{\xmlgrab}{}
15:
16: \XMLelement{docTitle}{}{\xmlgrab}{\title{#1}}
17: \XMLelement{docDate}{}{\xmlgrab}{\date{#1}}
18: \XMLelement{docAuthor}{}{\xmlgrab}{\author{#1}}
19:
20: \XMLelement{front}{}{}{}
21: \XMLelement{titlePage}{}{}{\maketitle}
22:
23: \newcounter{Sdiv}
24: \XMLelement{div1}{}{\setcounter{Sdiv}{1}}{}
25: \XMLelement{div2}{}{\setcounter{Sdiv}{2}}{}
26:
27: \XMLelement{head}{}{\xmlgrab}
28:   {\ifthenelse{\value{Sdiv} = 1}{\section{#1}}{}
29:    \ifthenelse{\value{Sdiv} = 2}{\subsection{#1}}{}}
30:
31: \XMLelement{hi}{\XMLattribute{rend}{\hirend}{}}
32:   {\ifthenelse{\equal{\hirend}{tt}}{\ttfamily}
33:    {\ifthenelse{\equal{\hirend}{bf}}{\bfseries}{\itshape}}}
34:   {}
35:
36: \XMLelement{lg}{}{\begin{verse}}{\end{verse}}
37: \XMLelement{l}{\XMLattribute{rend}{\rend}{}}
38:   {\xmlgrab}
39:   {\ifthenelse{\equal{\rend}{indent}}{\quad}{}#1\\}
```

Ce fichier contient essentiellement une série de commandes `\XMLelement`, dont la syntaxe est détaillée à la section B.1. Cette commande spécifie comment TEX compose un élément XML donné. Pour illustrer son fonctionnement, considérons les lignes 37 à 39, où les quatre paramètres de la commande `\XMLelement` sont utilisés. Le premier argument est le nom de l'élément à traiter (`<l>`). Le deuxième argument permet le traitement des attributs à l'aide de la commande

`\XMLattribute`, qui a trois arguments : le nom de l'attribut, la commande TEX qui contiendra la valeur de l'attribut s'il est utilisé et une valeur par défaut (ici nous sauvegardons la valeur de l'attribut `rend` dans la commande `\rend` et la valeur par défaut est vide). Le troisième argument de `\XMLelement` est la liste des commandes TEX à exécuter lorsqu'on rencontre la balise de début de l'élément (ici la commande `\xmlgrab` indique que nous désirons sauvegarder le contenu de l'élément). Finalement, le quatrième argument donne la liste des commandes TEX à exécuter lorsqu'on rencontre la balise de fin de l'élément. Ici nous vérifions si la valeur de l'attribut `rend` transmise dans la commande `\rend` est égale à `"indent"`, auquel cas on commence la ligne par un renfoncement d'une valeur d'un cadratin. Puis, dans tous les cas, le contenu de l'élément est récupéré via la variable TEX `#1` où la commande `\xmlgrab` l'a sauvegardé. Finalement la ligne est terminée avec la commande `\\`.

Ainsi, chacune des commandes `\XMLelement` spécifie en détail quelles actions TEX doit prendre lorsque `xmltex` rencontre l'élément en question dans le document source. Par exemple les lignes 6 à 12 traitent l'élément racine `TEI.2`. Pour sa balise de début nous générons le préambule LATEX (lignes 7–11), pour sa balise de fin nous terminons le document (ligne 12).

Le résultat du traitement du fichier `verlainetei.xml` avec `xmltex` est montré dans la partie supérieure (A) de la figure 1.

3. PassiveTEX : composer un fichier XML en utilisant les objets de formatage XSL

Une deuxième façon de composer un fichier XML avec TEX est de transformer le document source XML d'abord en un autre document XML contenant des objets de formatage XSL (voir section 3.1) en utilisant des feuilles de style XSLT, puis de composer ce document avec `xmltex` et PassiveTEX (voir section 3.2). Les deux étapes envisagées sont explicitées ci-dessous en utilisant le processeur `xt` de James Clark ([5], voir aussi la section 7.6.4 de [9]) :

```
xt fichier.xml feuille-de-style.xsl fotex.xml
latex fotex.tex
```

Le fichier `feuille-de-style.xsl` contient les commandes XSLT pour transformer un document XML balisé d'après la DTD utilisée pour le document XML à l'entrée `fichier.xml` en objets de formatage XSL. Le document XML résultant est écrit dans le fichier `fotex.xml`, qui est traité après par LATEX qui fera appel à PassiveTEX. Dans les sections suivantes nous décrirons plus en détail les objets de formatage XSL et PassiveTEX.

3.1. Les objets de formatage XSL

Un des principes de base de la spécification définissant les objets de formatage XSL [25] (XSL-OF) est qu'elle doit avoir un contenu sémantique assez riche pour englober les modèles de la norme DSSSL [13] et de CSS [19] (voir aussi [6]). XSL-OF est un langage de formatage abstrait, qui doit permettre de représenter plusieurs formats : imprimante, écran, audio, etc. Dans cet article nous ne nous intéresserons qu'à une sortie typographiquement de haute qualité (voir la section 6.3.2 de [7] qui traite de la création d'une toile de fichiers HTML ou plus généralement [9]). Le langage XSL-OF s'intègre de façon optimale au langage de transformation XSLT. Comme il s'agit d'un langage XML il offre un support sans faille pour l'internationalisation (différentes directions de composition dans un même paragraphe, tous les caractères Unicode).

Le langage XSL-OF représente le document formaté sous forme d'un arbre XML contenant les éléments suivants : séquences, modèles et régions de page, textes en bloc (paragraphes) et en ligne (à l'intérieur d'un paragraphe), caractères, tables, listes, liens, éléments flottants, notes de bas de page, inclusions graphiques, etc. Cet arbre peut être sérialisé comme une séquence d'éléments XML. Chacun des éléments XSL-OF peut avoir des attributs caractérisant plus en détail sa représentation, par exemple : couleur, positionnement, présence d'un cadre, espacements, police, césure, tabulatation.

Voici une partie du début du fichier `fotex.xml` qui correspond à la première page du recueil des poèmes de Verlaine après traitement par les feuilles de style XSLT pour la TEI [16].

```
 1: <fo:block font-size="16pt">
 2:  <fo:inline-sequence font-weight="bold">
 3:   Recueil de poèmes de Paul Verlaine
 4:  </fo:inline-sequence>
 5: </fo:block>
 6:
 7: <fo:block font-size="14pt">
 8:  <fo:inline-sequence font-style="italic">
 9:   Michel Goossens
10:  </fo:inline-sequence>
11: </fo:block>
12:
13: <fo:block font-size="14pt">Octobre 1999</fo:block>
14:
15: <fo:flow font-family="Times Roman" font-size="10pt">
16:  <fo:block keep-with-next="true" id="N104" text-align="start"
17:            font-size="14pt" text-indent="-3em" font-weight="bold"
18:            space-after="3pt" space-before.optimum="9pt">
19:   1. Poèmes saturniens (1866)
20:  </fo:block>
21:
```

```
22:  <fo:block keep-with-next="true" id="N116" text-align="start"
23:            font-size="12pt" font-weight="bold" space-after="2pt"
24:            space-before.optimum="4pt" text-indent="-3em">
25:    1.1.
26:     <fo:inline-sequence font-style="italic">
27:      Soleils couchants
28:     </fo:inline-sequence>
29:  </fo:block>
30:
31:  <fo:block text-align="start" space-before.optimum="4pt"
32:            space-after.optimum="4pt">
33:    <fo:block space-before.optimum="0pt" space-after.optimum="0pt">
34:      Une aube affaiblie
35:    </fo:block>
36:
37:   ...
38:  </fo:block>
39: </fo:block>
40:
41:   ...
42: </fo flow>
```

Les lignes 1 à 5 définissent un bloc de texte composé en 16 points gras (le titre du document), les lignes 7 à 11 correspondent à un bloc composé en 14 points italique (l'auteur), la ligne 13, finalement, est en 14 points romain (la date). Les lignes 15 à 20 décrivent un bloc pour le titre du premier niveau (le nom du recueil), puis les lignes 22 à 29 font de même pour le titre de deuxième niveau (le nom du poème). La numérotation des titres est générée automatiquement par le processeur XSLT. Une description détaillée des différents éléments XSL-OF et de leurs attributs se trouve dans la recommandation W3C [25].

Les feuilles de style XSLT `teixsl` [16] contiennent des déclarations qui pour chaque élément source indiquent comment il est transformé en un objet de formatage XSL (voir la section 7.6.6 de [9] pour une discussion plus détaillée). Ci-dessous nous montrons, par exemple, le traitement prévu pour un paragraphe (élément `p` aux lignes 1 à 6) et un texte en emphase (l'élément `emph` aux lignes 7 à 11).

```
 1: <xsl:template match="p">
 2:   <fo:block indent-start="10pt"
 3:             space-before="12pt">
 4:     <xsl:apply-templates/>
 5:   </fo:block>
 6: </xsl:template>
 7: <xsl:template match="emph">
 8:   <fo:inline-sequence font-style="italic">
 9:     <xsl:apply-templates/>
10:   </fo:inline-sequence>
11: </xsl:template>
```

3.2. PassiveTEX

PassiveTEX est une bibliothèque de macros TEX écrite par Sebastian Rahtz [15]. Elle permet de traiter un document XML contenant des objets de formatage XSL-OF.

Puisque PassiveTEX est basé sur TEX il offre un environnement de développement idéal pour l'expérimentation avec les objets de formatage XSL-OF en se basant sur un moteur de composition stable, bien connu et disponible partout.

PassiveTEX n'utilise pas les commandes de haut niveau comme LATEX. Le langage XSL-OF qui gère sections, listes, références croisées, bibliographie, etc.

XSL-OF est un langage XML. Son codage de base est donc Unicode et par défaut toutes les entités sont mappées sur leur positions Unicode. Tous les espaces verticaux et horizontaux doivent être explicités, seules les coupures de ligne et de page sont sous le contrôle de TEX, le reste devant être spécifié par l'utilisateur dans la feuille de style XSLT qui génère les objets XSL-FO.

En utilisant PassiveTEX avec pdfTEX, une variante de TEX, on peut obtenir diectement une sortie PDF sans étapes intermédiaires. Il est à souligner que l'utilisation du moteur TEX est tout à fait transparent et en général on ne se rend même pas compte que TEX est appelé.

3.2.1. Avantages et désavantages de l'approche PassiveTEX

Le fait que PassiveTEX soit basé sur TEX a quelques côtés positifs. Il permet le développement rapide d'un prototype pour expérimenter avec des variations pour la représentation d'un élément source donné. TEX gère les polices, les inclusions graphiques, la gestion des hyperliens, etc. Grâce au mécanisme de césure de TEX différentes langues peuvent être traitées correctement. Les formules mathématiques seront parfaitement composées par TEX.

Du côté négatif nous mentionnons que PassiveTEX reste toujours lié au modèle de mise en page TEX. Ainsi les élément XSL-OF sont forcés à s'adapter au modèle TEX. Avec LATEX, qui est déjà un langage de balisage de haut niveau, on utilise trop facilement et sans le savoir des valeurs par défaut de LATEX pour la composition de certains éléments abstraits et une telle correspondance n'est souvent pas souhaitable.

À ces considérations techniques il faut ajouter le fait que l'utilisateur-type du web ne connait souvent rien ou très peu du langage TEX et ses macros lui semblent obscures, trop complexes et non-intuitives. Finalement, TEX est un programme monolithique et il est difficile de le modulariser ou de l'inclure dans d'autres applications ce qui n'en fait pas un outil idéal dans le contexte du web.

3.2.2. Support des objets de formatage XSL-OF

La version actuelle de PassiveTEX inclut un support presque complet pour les caractéristiques XSL-OF suivantes :

— les boîtes de type paragraphe (`blocks`) ;
— les éléments à l'intérieur d'un paragraphe (`inline sequences`) ;
— les listes ;
— l'inclusion de graphiques ;
— les éléments flottants ;
— les propriétés des polices ;
— les liens (`links`).

Les caractéristiques suivantes ne sont que très partiellement réalisées :

— les spécifications de la page ;
— les tables et leur caractéristiques ;
— les marges (`margin`), les limites (`border`) et les zones de remplissage (`padding`) ;
— la pagination et les caractéristiques de la page courante.

Enfin, actuellement il n'y a pas encore de support pour :

— la composition multi-directionnelle (`bidi`) ;
— les fonds (`background`) ;
— les caractéristiques audio ;
— un grand nombre d'autres caractéristiques spécifiques, en particulier la notation abrégée du type `font="Times Roman 11pt bold"` n'est pas reconnue.

3.2.3. PassiveTEX et les mathématiques

Pour le codage mathématique PassiveTEX traite les éléments du langage XML MathML [20] directement. Dans la feuille de style XSLT qui transforme le document source XML en objets de formatage les éléments `<math>` et leurs descendants sont transmis inchangés (pour le code exacte voir les lignes 19 à 31 de la feuille de style `foplus` de la section 5.4).

Le traitement des éléments MathML est intégré à `xmltex` à l'aide d'un fichier `mathml2.xml`. Le nom du fichier est spécifié dans un fichier de configuration. Par exemple le fichier `xmltex.cfg` contient la ligne suivante :

```
\NAMESPACE{http://www.w3.org/1998/Math/MathML}          {mathml2.xmt}
```

Ainsi, si un fichier source fait référence au domaine nominal MathML spécifié le fichier `mathml2.xml` sera chargé.

Pour avoir une idée de la façon dont `xmltex` traite les mathématiques nous reproduisons ci-dessous quelques lignes du contenu du fichier `mathml2.xmt`.

```
 1: \DeclareNamespace{m}{http://www.w3.org/1998/Math/MathML}
 2:
 3: \XMLname{formula}{\FORMULA}
 4:
 5: \XMLelement{m:math}
 6:   {}
 7:   {\ifthenelse{\equal{\XML@parent}{\FORMULA}}{}{\[}}
 8:   {\ifthenelse{\equal{\XML@parent}{\FORMULA}}{}{\]}}
 9:
10: \XMLelement{m:mn}
11:   {}
12:   {\xmlgrab}
13:   {\mathrm{#1}}
14:
15: \XMLelement{m:mfrac}
16:   {}
17:   {\xmlgrab}
18:   {\xmltextwochildren\frac{}#1}
19:
20: \XMLelement{m:msqrt}
21:   {}
22:   {\xmlgrab}
23:   {\sqrt{#1}}
```

La ligne 3 définit la commande `\FORMULA`, qui contiendra le nom littéral `formula` précédé du préfixe du domaine nominal interne généré par `xmltex`, ce qui permettra de le comparer à un nom d'élément à l'intérieur d'un document. En particulier, aux lignes 7 et 8 nous l'utilisons pour vérifier si le nom du parent de l'élément `m:math` (ligne 5) est `formula`, auquel cas le mode mathématique est déjà géré. Dans le cas contraire nous devons commencer (`\[` à la ligne 7) un environnment mathématique LaTeX et le terminer (`\]` à la ligne 8) lorsque nous rencontrons la balise de début et de fin, respectivement. Les lignes 10–13 traitent les noms mathématiques, les lignes 15–18 l'élément `frac` (fraction) et les lignes 20–23 l'élément `sqrt` (racine carrée). Plus généralement, le fichier `mathml2.xmt` contient du code pour traiter tous les éléments de la syntaxe de présentation MathML. Nous verrons à la section 5 quelques exemples de formules mathématiques traitées par `xmltex`.

3.2.4. Développement futur de PassiveTEX

Comme déjà évoqué à la section 3.2.2, il reste encore pas mal de travail à PassiveTEX avant qu'il ne couvre plus complètement les différents éléments du

langage XSL-OF. Dans un futur proche les efforts vont se concentrer sur les domaines suivants :

— le support du langage MathML ;
— traiter plus de caractéristiques XSL-OF (couleurs, polices) ;
— les tables complexes ;
— les éléments graphiques SVG [22]. Il existe plusieurs approches :
  – interprétation directe et mapping sur les primitives PDF ;
  – transformation en MetaPost [11] et appel dynamique au processeur MetaPost ;
  – traitement prélimininaire pour générer du code dans un langage graphique TEX existant.

  Dans tous les cas les fragments codés en SVG devront être interprétés directement pour permettre l'exécution des fonctions graphiques (comme composer un texte à un angle donné).
— le traitement de langues non-latines, où une approche prometteuse semble l'utilisation du moteur TEX 16 bits [10], dont Unicode est le codage interne.

3.3. Traitement de l'exemple TEI avec XSLT et PassiveTEX

À la section 3 nous avons montré les deux étapes pour composer un fichier XML avec XSLT et PassiveTEX. Pour l'exemple de la TEI qui nous intéresse ici nous utilisons des feuilles de style XSLT pour la TEI [16] développés par Sebastian Rahtz.

```
xt verlainetei.xml tei.xsl fotex.xml latex fotex.tex
```

Le fichier XML à l'entrée `verlainetei.xml` est celui que nous avons utilisé précédemment à la section 2.1, `tei.xsl` contient les transformation XSLT pour transformer un document XML balisé d'après la DTD TEI en objets de formatage XSL. Le résultat est le document XML `fotex.xml`. Il contient les objets de formatage XSL et il est traité avec LATEX (qui appelera PassiveTEX et `xmltex`) par l'intermédiaire du fichier `fotex.tex` qui contient :

```
1: \def\xmlfile{fotex.xml}
2:   \input xmltex.tex
3:   \end{document}
```

Le résultat du traitement du fichier `verlainetei.xml` avec la procédure décrite ci-dessus est montré dans la partie centrale (B) de la figure 1.

A côté de PassiveTEX il y a FOP [2], à l'origine développé par James TAUBER, mais récemment repris par le projet Apache. FOP, une application écrite en Java, transforme les objets de formatage XSL-OF en PDF. FOP est écrit en Java et est donc très portable. La version actuelle couvre déjà une grande partie du langage XSL-OF, mais dans plusieurs domaines FOP n'a pas encore atteint la qualité typographique de TEX. Des réalisations commerciales pour composer des documents XML contenant de objets de formatage XSL-OF (*RenderX*, *Epic* d'ArborText) sont annoncées mais pas encore disponibles.

4. Composer un fichier XML en le transformant en LATEX avec une feuille de style XSLT

Une troisième approche pour composer un document XML avec TEX est de le transformer directement en LATEX à l'aide d'une feuille de style XSLT développée *ad hoc* (voir les sections 5.2 et 6 de [7] et la norme XSLT [24]).

Une feuille de style XSLT, qui s'inspire beaucoup de celle utilisée pour piloter la composition du recueil des poèmes de Verlaine à la section 2.2 est montrée ci-dessous.

```
 1: <?xml version='1.0' encoding="ISO-8859-1"?>
 2: <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 3:                 version="1.0"
 4:                 xmlns="http://www.tug.org/LaTeX">
 5:
 6: <xsl:output method="text" indent="no" encoding="ISO-8859-1"/>
 7: <xsl:strip-space elements="*"/>
 8:
 9: <xsl:template match="TEI.2">
10: \documentclass{article}
11: \usepackage{a4wide}
12: \usepackage[T1]{fontenc}
13: \begin{document}
14: <xsl:apply-templates/>
15: \end{document}
16: </xsl:template>
17:
18: <xsl:template match="teiHeader"/>
19: <xsl:template match="text|front|body">
20: <xsl:apply-templates/>
21: </xsl:template>
22:
23: <xsl:template match="titlePage">
24: \title{<xsl:value-of select="docTitle/titlePart"/>}
25: \author{<xsl:value-of select="docAuthor"/>}
26: \date{<xsl:value-of select="docDate"/>}
```

```
27: \maketitle
28: </xsl:template>
29:
30: <xsl:template match="div1/head">
31: \section{<xsl:apply-templates/>}
32: </xsl:template>
33:
34: <xsl:template match="div2/head">
35: \subsection{<xsl:apply-templates/>}
36: </xsl:template>
37:
38: <xsl:template match="lg">
39: \begin{verse}
40: <xsl:apply-templates/>
41: \end{verse}
42: </xsl:template>
43:
44: <xsl:template match="l">
45: <xsl:if test="@rend='indent'">\quad </xsl:if>
46: <xsl:apply-templates/>\\
47: </xsl:template>
48:
49: <xsl:template match="hi">
50: <xsl:choose>
51: <xsl:when test="@rend='tt'">{\ttfamily </xsl:when>
52: <xsl:when test="@rend='bf'">{\ttseries </xsl:when>
53: <xsl:otherwise>{\itshape </xsl:otherwise>
54: </xsl:choose>
55: <xsl:apply-templates/>}
56: </xsl:template>
57:
58: </xsl:stylesheet>
```

Pour la plupart des commandes `\XMLelement` du fichier `verlaine.xmt` de la section 2.2 on trouve ici une instruction `<xsl:template match="...">`. Le motif spécifié par l'attribut `match` choisit l'élément dans l'arbre source qui sera transformé d'après le modèle spécifié. Ainsi les lignes 9–16 (élément racine `TEI.2`) correspondent aux lignes 6–12 dans le fichier `xmt` ; la ligne 18 est l'équivalent de la ligne 14 (dans les deux cas le contenu de l'élément `teiHeader` est ignoré) ; les lignes 23–28 (élément `titlePage`) correspondent aux lignes 16–18 et 21 ; les lignes 30–36 (élément `head` descendant d'un élément `div1` ou `div2`) qui s'occupent du traitement des sections et de leurs titres correspondent aux lignes 23–29. Et ainsi de suite pour les éléments `lg` (lignes 38–42 et ligne 37), `l` (lignes 44–47 et 37–39), `hi` (lignes 49–56 et 31–34).

En présentant cette feuille de style XSLT `verlaineteitex.xsl` et le fichier source XML `verlainetei.xml` à un processeur XSLT nous obtenons un fichier TEX `verlaineteitex.tex` comme suit :

```
xt verlainetei.xml verlaineteitex.xsl verlaineteitex.tex
```

En traitant le fichier `verlaineteitex.tex` avec LaTeX et après génération d'une sortie PostScript, nous obtenons les trois pages montrées à la partie inférieure (C) de la figure 1. La ressemblance pratiquement parfaite avec le sortie générée à l'aide de `xmltex` (A) montre l'équivalence de ces deux approches.

5. DocBook et les mathématiques

Jusqu'ici nous nous sommes intéressés à des documents saisis en utilisant un balisage basé sur la DTD TEI. Dans cette section nous montrons que les mêmes techniques pour composer un document XML avec TeX sont valables pour d'autres DTD et nous utiliserons comme exemple *DocBooc* [18] et MathML [20].

5.1. Le modèle DocBook

DocBook [18] propose un modèle XML spécialement adapté au balisage de documents techniques, en particulier dans les domaines des équipements d'ordinateurs ou de logiciels informatiques.

La DTD Docbook propose des centaines d'éléments pour baliser le plus clairement et explicitement possible les différentes composantes d'un document (livre, manuel, article, etc.), non seulement au niveau hiérarchique, mais également sémantique. La structure de la DTD a été optimisée pour permettre sa personnalisation. Ainsi il est relativement aisé d'ajouter ou d'enlever des éléments ou des attributs, de changer le contenu de certains groupes structurels d'éléments ou de restreindre les valeurs de certains attributs.

Norman Walsh a écrit des feuilles de style XSL qui transforment les documents XML balisé d'après la DTD DocBook en HTML ou en objets de formatage XSL. C'est cette deuxième alternative que nous étudierons afin d'obtenir une sortie PDF ou PostScript avec PassiveTeX et `xmltex`.

Nous n'allons pas donner beaucoup de détails sur le langage DocBook. Ci dessous nous montrons le début du fichier `testmml.xml`, que nous avons créé pour tester les différentes approches introduites dans cet article avec la DTD DocBook. Le sous-ensemble interne de la DTD (lignes 4–14) contient la définition de quelques entités générales (lignes 4–8), entités paramètres (lignes 9 à 11), puis nous déclarons l'attribut `xmlns` de l'élément `math` (lignes 12–13) pour permettre de spécifier le domaine nominal de MathML. Finalement (ligne 14) nous chargeons par un appel d'entité paramètre (défini à la ligne 11) la DTD du langage MathML. Les autres lignes sont assez faciles à comprendre. Nous reviendrons sur l'inclusion des mathématiques plus tard.

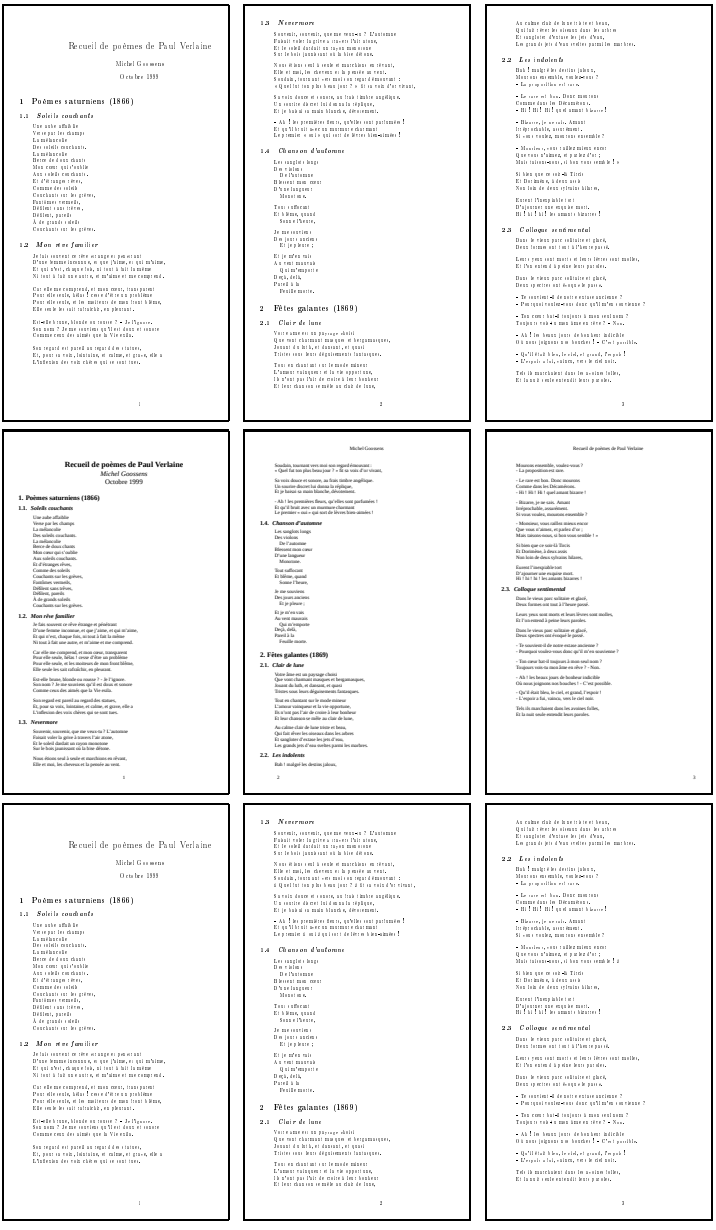


FIGURE 1: Trois approches pour le fichier `verlainetei.xml` :
A) `xmltex` seul (section 2) ; B) transformation XSLT en objets de formatage et PassiveTEX (section 3) ; C) transformation XSLT en LATEX (section 4).

```
 1: <?xml version="1.0" encoding="ISO-8859-1"?>
 2: <!DOCTYPE article SYSTEM
 3: "/usr/local/share/docbookxml/3.17/docbookx.dtd" [
 4:  <!ENTITY oelig "&#x0153;">
 5:  <!ENTITY LaTeX "LaTeX">
 6:  <!ENTITY TeX   "TeX">
 7:  <!ENTITY PTeX  "PassiveTeX">
 8:  <!ENTITY xmltex "<application>xmltex</application>">
 9:  <!ENTITY % equation.content "(math+)">
10:  <!ENTITY % inlineequation.content "(math+)">
11:  <!ENTITY % mathml SYSTEM "mathmldtd/mathml.dtd">
12:  <!ATTLIST math xmlns CDATA #FIXED
13:                    "http://www.w3.org/1998/Math/MathML">
14: %mathml; <!- load MathML ->
15: ]>
16: <article>
17: <artheader>
18: <title>Un document Docbook avec quelques formules</title>
19: <author><firstname>Michel</firstname> <surname>Goossens</surname>
20: </author>
21: <pubdate>Mardi, le 28 mars 2000</pubdate>
22: <abstract>
23: <para>
24: Ce document XML est balisé d'après le modèle DocBook. Il montre
25: les éléments principaux, ainsi que quelques exemples d'expressions
26: mathématiques où nous utilisons le balisage MathML.
27: </para>
28: </abstract>
29: </artheader>
30: <section>
31: <title>Le modèle DocBook</title>
32: <para>
33: DocBook <xref role="bib" linkend="docbook" endterm="docbookab"/>
34: propose un modèle XML <xref role="bib" linkend="mgxml"
35: endterm="mgxmlab"/> pour baliser les documents techniques,
36: spécialement dans les domaines des équipements d'ordinateurs ou de
37: logiciels informatiques.
38: </para>
```

5.2. MathML et les mathématiques

MathML [20] est une recommandation W3C pour coder l'information mathématique afin de faciliter la communication dans l'éducation et les sciences.

MathML consiste en deux parties : présentation (forme) et sémantique (contenu). L'information mathématique codée en MathML peut être représentée sous plusieurs formes : écran graphique, synthétiseur vocal, programmes d'algèbre par ordinateur, autres langages de balise mathématiques (comme TEX), terminal texte, imprimantes (PostScript), sortie braille, etc.

MathML permet la représentation d'expressions longues et complexes et offre un mécanisme d'extension. Le code source MathML est facile à générer par une application et à éditer (même si la syntaxe d'une expression MathML peut paraître inutilement longue et complexe à l'œil humain).

5.2.1. Le balisage de présentation

La partie *présentation* de MathML décrit la disposition dans le plan des différents éléments d'une expression mathématique. MathML contient une trentaine d'éléments de présentation, qui décrivent la structure hiérarchique de l'expression par rapport à un modèle classique visuel. Une cinquantaine d'attributs permettent un contrôle micro-typographique de ces éléments. Ce type de balisage est assez proche de la syntaxe de TEX et il est traité par le fichier `mathml2.xmt` de `xmltex` décrit à la section 3.2.3.

5.2.2. Le balisage sémantique

La structure sémantique d'une expression mathématique est nécessaire pour évaluer sa valeur ou pour l'utiliser dans des calculs. Sa représentation visuelle (le balisage de présentation) ne contient pas toujours toute l'information requise. Ainsi MathML propose un balisage du *contenu* d'une expression.

Le balisage sémantique décrit les objets mathématiques et permet un codage direct de l'arbre structurel d'une expression mathématique. Un certain nombre d'éléments de base ont une sémantique par défaut, mais il existe un mécanisme pour associer une sémantique spécifique à un élément donné.

Les éléments sémantiques MathML se groupent en quelques catégories de base : les conteneurs, les opérateurs et fonctions, les identificateurs, les relations, les conditions, les mappings sémantiques, les constantes et symboles. Les expressions sémantiques MathML sont construites en utilisant ces blocs.

Ci-dessous est le balisage sémantique d'une équation matricielle. Son balisage de présentation se trouve aux lignes 21–36 de l'exemple de la section 5.3.

```
 1: <reln>
 2:   <eq/>
 3:   <ci>A</ci>
 4:   <matrix>
 5:     <matrixrow>
 6:       <ci>x</ci><ci>y</ci>
 7:     </matrixrow>
 8:     <matrixrow>
 9:       <ci>z</ci><ci>w</ci>
10:     </matrixrow>
11:   </matrix>
12: </reln>
```

5.3. Traiter un document DocBook avec `xmltex`

Le fichier `testmml` introduit à la section 5.1 contient deux exemples de balisage MathML de présentation. Ils sont inclus dans un élément mathématique DocBook (pour la TEI il existe un élément `formula` comme conteneur de balisage mathématique). Ainsi les lignes 6 à 10 montrent une formule mathématique à l'intérieur d'un paragraphe balisé par l'élément `inlineequation`. Les éléments MathML sont entourés d'un élément `math` qui est déclaré dans la DTD MathML, qui est chargé dans le préambule du document Docbook (ligne 14 du document `testmml.xml` de la section 5.1). Son domaine nominal y est spécifié aux lignes 12 et 13. Cela signifie que lorsque `xmltex` rencontre un élément `math` il l'associe à ce domaine nominal et il chargera le fichier `mathml2.xmt` pour traiter les éléments XML contenus dans l'élément `math` en question (voir la section 3.2.3). D'une façon analogue les lignes 19–39 montrent un élément `informalequation`, qui entoure une équation mathématique mise en évidence. Ici aussi `xmltex` se limite à transmettre le contenu de l'élément `math` au fichier `mathml2.xmt` qui traitera les mathématiques directement en TEX.

```
 1: <section>
 2: <title>Une exemple de MathML</title>
 3: <para>
 4: Une formule MathML peut être composée à l'intérieur d'une ligne,
 5: comme ici
 6: <inlineequation>
 7: <math>
 8: <mi>E</mi><mo>=</mo><mi>m</mi><msup><mi>c</mi><mn>2</mn></msup>
 9: </math>
10: </inlineequation>, la fameuse équation d'Einstein.
11: </para>
12:
13: <para>
14: Une équation peut aussi être mise en évidence en utilisant l'élément
15: <sgmltag class="element">informalequation</sgmltag>, comme montré
16: dans l'exemple ci-dessous contenant une matrice :
17: </para>
18:
19: <informalequation>
20: <math>
21: <mrow>
22:   <mi>A</mi>
23:   <mo>=</mo>
24:   <mfenced open="[" close="]">
25:     <mtable><!- table ou matrice     ->
26:       <mtr> <!- ligne dans une table ->
27:          <mtd><mi>x</mi></mtd><!- table  ->
28:          <mtd><mi>y</mi></mtd><!- entrée ->
29:       </mtr>
30:       <mtr>
31:          <mtd><mi>z</mi></mtd>
```

```
32:          <mtd><mi>w</mi></mtd>
33:       </mtr>
34:     </mtable>
35:   </mfenced>
36: </mrow>
37: <mtext>.</mtext>
38: </math>
39: </informalequation>
40: </section>
41: </section>
```

Pour traiter le fichier `textmml.xml` directement avec `xmltex` nous avons créé un fichier TEX `textmml.tex` qui contient :

```
1: \def\xmlfile{testmml.xml}
2:   \input xmltex.tex
3:   \end{document}
```

Nous avons aussi généré un fichier de configuration `textmml.cfg`, qui contient les lignes suivantes :

```
1: \SYSTEM{/usr/local/share/docbookxml/3.17/docbookx.dtd} {docbook.xmt}
2: \NAME{article}                                         {docbook.xmt}
3: \NAME{book}                                            {docbook.xmt}
4: \NAMESPACE{http://www.w3.org/1998/Math/MathML}         {mathml2.xmt}
```

Les lignes 1 à 3 indiquent à `xmltex` qu'il doit charger le fichier `docbook.xmt` quand il rencontre l'identificateur système indiqué à la ligne 1 ou la balise de début de l'élément `article` ou `book`. La ligne 4 informe `xmltex` qu'il doit charger le fichier `mathml2.xmt` quand il rencontre un élément qui fait référence au domaine nominal MathML indiqué.

Le contenu du fichier `docbook.xmt`, qui contient des instructions TEX de type `\XMLelement` pour associer du code TEX aux éléments XML de DocBook dans le document source, est similaire au fichier `verlaine.xmt` dont nous avons parlé pour la DTD TEI dans le contexte du recueil des poèmes de Verlaine à la section 2.2. Il est donc inutile de le disséquer en détail ici. Nous mentionnons cependant que sa structure est un peu plus complexe parce que plusieurs éléments de la DTD DocBook peuvent avoir divers types d'éléments comme parent, auquel cas nous devons tester le nom de ce dernier. Voici un exemple pour l'élément `title`, dont le traitement du contenu varie d'après le type de l'élément parent (les commandes de type `\ARTHEADER`, `\SECTION`, etc., contiennent les noms qualifiés des éléments `artheader`, `section`, etc.).

```
1: \XMLelement{title}{}
2:   {\xmlgrab}
3:   {\ifthenelse{\equal{\XML@parent}{\ARTHEADER}}{\title{#1}}{}
```

```
  \ifthenelse{\equal{\XML@parent}{\SECTION}}
  {\protected@xdef\temp{
   \expandafter\noexpand\ifcase\SCOUNT
   \section\or\subsection\or\subsubsection\fi
   {#1}\ifx\idval\@nnil\else\noexpand\label{\idval}\fi}
   \aftergroup\temp}{}%%% fin de parents de type "section"
  \ifthenelse{\equal{\XML@parent}{\FIGURE}}{\caption{#1}}{}
  \ifthenelse{\equal{\XML@parent}{\TABLE}}{\caption{#1}}{}
  \ifthenelse{\equal{\XML@parent}{\BIBLIOGRAPHY}}{}{}%%% rien faire
  \ifthenelse{\equal{\XML@parent}{\BIBLIOSET}}
   {\ifthenelse{\equal{\BIBLIOSETRELATION}{journal}}
       {« #1 »}{\emph{#1}}}{}
  \ifthenelse{\equal{\XML@parent}{\BIBLIOENTRY}}{\emph{#1}}{}
 }
```

Le résultat du traitement du fichier `textmml.xml` avec `xmltex` en utilisant le fichier `testmml.tex` et `docbook.xmt` est montré à la figure 2.

5.4. Traiter un document DocBook avec XSLT et PassiveTEX

Nous pouvons également transformer les éléments XML en objets de formatage XSL et puis utiliser PassiveTEX, comme évoqué à la section 3 dans le contexte de la DTD TEI. Pour la DTD DocBook nous utiliserons les feuilles de styles XSLT développées par Norman Walsh [18] pour obtenir un fichier `fotex` à traiter par LaTeX et PassiveTEX. Pour permettre l'inclusion du traitement des éléments mathématiques balisé en MathML, nous avons du ajouter quelques instructions XSLT à celles définies dans les feuilles de style de DocBook distribuées par Norman Walsh. Voici le fichier `foplus.xsl` qui nous avons créé.

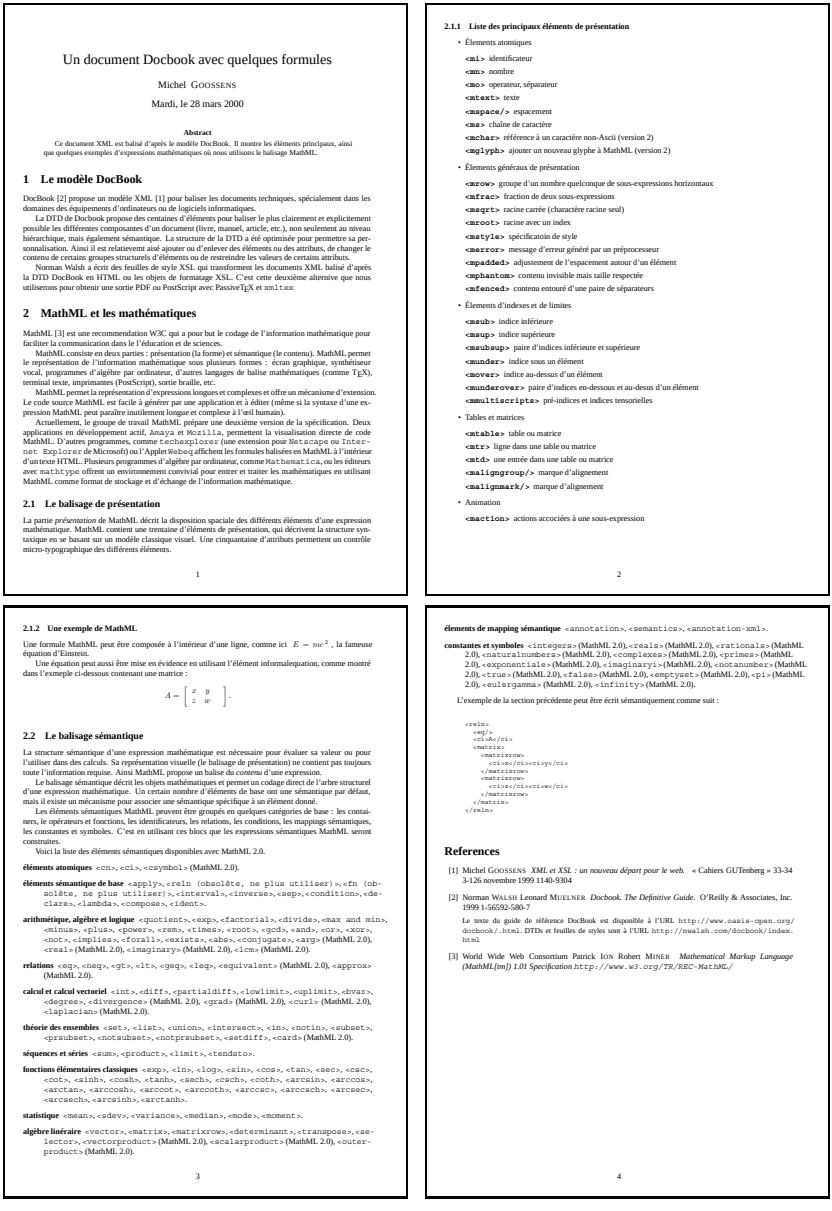
```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                xmlns:m="http://www.w3.org/1998/Math/MathML">

<xsl:import href="~/docbookxsl/1.09/fo/docbook.xsl"/>

<!- ************************************************************
       Suppléments de Michel aux feuilles de style XSL-OF de Norm
      ************************************************************ ->

<xsl:template match="informalequation|inlineequation">
<xsl:element name="{name(.)}">
 <xsl:apply-templates/>
</xsl:element>
</xsl:template>

```



Un document Docbook avec quelques formules

Michel GOOSSENS

Mardi, le 28 mars 2000

Abstract

Ce document XML est balisé d'après le modèle DocBook. Il montre les éléments principaux, ainsi que quelques exemples d'expressions mathématiques où nous utilisons le balisage MathML.

1 Le modèle DocBook

DocBook [2] propose un modèle XML [1] pour baliser les documents techniques, spécialement dans les domaines des équipements d'ordinateurs ou de logiciels informatiques.

La DTD de Docbook propose des centaines d'éléments pour baliser le plus clairement et explicitement possible les différentes composantes d'un document (livre, manuel, article, etc.), non seulement au niveau hiérarchique, mais également sémantique. La structure de la DTD a été optimisée pour permettre sa personnalisation. Ainsi il est relativement aisé ajouter ou d'enlever des éléments ou des attributs, de changer le contenu de certains groupes structurels d'éléments ou de restreindre les valeurs de certains attributs.

Norman Walsh a écrit des feuilles de style XSL qui transforment les documents XML balisé d'après la DTD DocBook en HTML ou les objets de formatage XSL. C'est cette deuxième alternive que nous utiliserons pour obtenir une sortie PDF ou PostScript avec PassiveTEX et `xmltex`

2 MathML et les mathématiques

MathML [3] est une recommendation W3C qui a pour but le codage de l'information mathématique pour faciliter la communication dans le l'éducation et de sciences.

MathML consiste en deux parties : présentation (la forme) et sémantique (le contenu). MathML permet le représentation de l'information mathématique sous plusieurs formes : écran graphique, synthétiseur vocal, programmes d'algèbre par ordinateur, d'autres langages de balise mathématiques (comme TEX), terminal texte, imprimantes (PostScript), sortie braille, etc.

MathML permet la représentation d'expressions longues et complexes et offre un mécanisme d'extension. Le code source MathML est facile à générer par une application et à éditer (même si la syntaxe d'une expression MathML peut paraître inutilement longue et complexe à l'œil humain).

Actuellement, le groupe de travail MathML prépare une deuxième version de la spécification. Deux applications en développement actif, `Amaya` et `Mozilla`, permettent la visualisation directe de code MathML. D'autres programmes, comme `techexplorer` (une extension pour `Netscape` ou `Internet Explorer` de Microsoft) ou l'Applet `Webeq` affichent les formules balisées en MathML à l'intérieur d'un texte HTML. Plusieurs programmes d'algèbre par ordinateur, comme `Mathematica`, ou les éditeurs avec `mathtype` offrent un environnement convivial pour entrer et traiter les mathématiques en utilisant MathML comme format de stockage et d'échange de l'information mathématique.

2.1 Le balisage de présentation

La partie *présentation* de MathML décrit la disposition spaciale des différents éléments d'une expression mathématique. MathML contient une trentaine d'éléments de présentation, qui décrivent la structure syntaxique en se basant sur un modèle classique visuel. Une cinquantaine d'attributs permettent un contrôle micro-typographique des différents éléments.

1

2.1.1 Liste des principaux éléments de présentation

- Éléments atomiques

  <mi> identificateur
  <mn> nombre
  <mo> operateur, séparateur
  <mtext> texte
  <mspace/> espacement
  <ms> chaîne de caractère
  <mchar> référence à un caractère non-Ascii (version 2)
  <mglyph> ajouter un nouveau glyphe à MathML (version 2)

- Éléments généraux de présentation

  <mrow> groupe d'un nombre quelconque de sous-expressions horizontaux
  <mfrac> fraction de deux sous-expressions
  <msqrt> racine carrée (charactère racine seul)
  <mroot> racine avec un index
  <mstyle> spécificatoin de style
  <merror> message d'erreur généré par un préprocesseur
  <mpadded> adjustement de l'espacement autour d'un élément
  <mphantom> contenu invisible mais taille respectée
  <mfenced> contenu entouré d'une paire de séparateurs

- Éléments d'indexes et de limites

  <msub> indice inférieure
  <msup> indice supérieure
  <msubsup> paire d'indices inférieure et supérieure
  <munder> indice sous un élément
  <mover> indice au-dessus d'un élément
  <munderover> paire d'indices en-dessous et au-dessus d'un élément
  <mmultiscripts> pré-indices et indices tensorielles

- Tables et matrices

  <mtable> table ou matrice
  <mtr> ligne dans une table ou matrice
  <mtd> une entrée dans une table ou matrice
  <malignmgroup/> marque d'alignement
  <malignmark/> marque d'alignement

- Animation

  <maction> actions accociées à une sous-expression

2

2.1.2 Une exemple de MathML

Une formule MathML peut être composée à l'intérieur d'une ligne, comme ici $E = mc^2$, la fameuse équation d'Einstein.

Une équation peut aussi être mise en évidence en utilisant l'élément informalequation, comme montré dans l'exemple ci-dessous contenant une matrice :

$$A = \begin{bmatrix} x & y \\ z & w \end{bmatrix}.$$

2.2 Le balisage sémantique

La structure sémantique d'une expression mathématique est nécessaire pour évaluer sa valeur ou pour l'utiliser dans des calculs. Sa représentation visuelle (le balisage de présentation) ne contient pas toujours toute l'information requise. Ainsi MathML propose un balise du *contenu* d'une expression.

Le balisage sémantique décrit les objets mathématiques et permet un codage direct de l'arbre structurel d'une expression mathématique. Un certain nombre d'éléments de base ont une sémantique par défaut, mais il existe un mécanisme pour associer une sémantique spécifique à un élément donné.

Les éléments sémantiques MathML peuvent être groupés en quelques catégories de base : les containers, le opérateurs et fonctions, les identificateurs, les relations, les conditions, les mappings sémantiques, les constantes et symboles. C'est en utilisant ces blocs que les expressions sémantiques MathML seront construites.

Voici la liste des éléments sémantiques disponibles avec MathML 2.0.

éléments atomiques <cn>, <ci>, <csymbol> (MathML 2.0).

éléments sémantique de base <apply>, <reln (obsolète, ne plus utiliser)>, <fn (obsolète, ne plus utiliser)>, <interval>, <inverse>, <sep>, <condition>, <declare>, <lambda>, <compose>, <ident>.

arithmétique, algèbre et logique <quotient>, <exp>, <factorial>, <divide>, <max and min>, <minus>, <plus>, <power>, <rem>, <times>, <root>, <gcd>, <and>, <or>, <xor>, <not>, <implies>, <forall>, <exists>, <abs>, <conjugate>, <arg> (MathML 2.0), <real> (MathML 2.0), <imaginary> (MathML 2.0), <lcm> (MathML 2.0).

relations <eq>, <neq>, <gt>, <lt>, <geq>, <leq>, <equivalent> (MathML 2.0), <approx> (MathML 2.0).

calcul et calcul vectoriel <int>, <diff>, <partialdiff>, <lowlimit>, <uplimit>, <bvar>, <degree>, <divergence> (MathML 2.0), <grad> (MathML 2.0), <curl> (MathML 2.0), <laplacian> (MathML 2.0).

théorie des ensembles <set>, <list>, <union>, <intersect>, <in>, <notin>, <subset>, <prsubset>, <notsubset>, <notprsubset>, <setdiff>, <card> (MathML 2.0).

séquences et séries <sum>, <product>, <limit>, <tendsto>.

fonctions élémentaires classiques <exp>, <ln>, <log>, <sin>, <cos>, <tan>, <sec>, <csc>, <cot>, <sinh>, <cosh>, <tanh>, <sech>, <csch>, <coth>, <arcsin>, <arccos>, <arctan>, <arccosh>, <arccot>, <arccoth>, <arccsc>, <arccsch>, <arcsec>, <arcsech>, <arcsinh>, <arctanh>.

statistique <mean>, <sdev>, <variance>, <median>, <mode>, <moment>.

algèbre linéraire <vector>, <matrix>, <matrixrow>, <determinant>, <transpose>, <selector>, <vectorproduct> (MathML 2.0), <scalarproduct> (MathML 2.0), <outerproduct> (MathML 2.0).

3

élements de mapping sémantique <annotation>, <semantics>, <annotation-xml>.

constantes et symboles <integers> (MathML 2.0), <reals> (MathML 2.0), <rationals> (MathML 2.0), <naturalnumbers> (MathML 2.0), <complexes> (MathML 2.0), <primes> (MathML 2.0), <exponentiale> (MathML 2.0), <imaginaryi> (MathML 2.0), <notanumber> (MathML 2.0), <true> (MathML 2.0), <false> (MathML 2.0), <emptyset> (MathML 2.0), <pi> (MathML 2.0), <eulergamma> (MathML 2.0), <infinity> (MathML 2.0).

L'exemple de la section précédente peut être écrit sémantiquement comme suit :

```
<reln>
  <eq/>
  <ci>A</ci>
  <matrix>
    <matrixrow>
      <ci>x</ci><ci>y</ci>
    </matrixrow>
    <matrixrow>
      <ci>z</ci><ci>w</ci>
    </matrixrow>
  </matrix>
</reln>
```

References

[1] Michel GOOSSENS *XML et XSL : un nouveau départ pour le web.* « Cahiers GUTenberg » 33-34 3-126 novembre 1999 1140-9304

[2] Norman WALSH Leonard MUELNER *Docbook. The Definitive Guide.* O'Reilly & Associates, Inc. 1999 1-56592-580-7

Le texte du guide de référence DocBook est disponible à l'URL `http://www.oasis-open.org/docbook/.html`. DTDs et feuilles de styles sont à l'URL `http://nwalsh.com/docbook/index.html`

[3] World Wide Web Consortium Patrick ION Robert MINER *Mathematical Markup Language (MathML[tm]) 1.01 Specification* `http://www.w3.org/TR/REC-MathML/`

4

FIGURE 2: Sortie générée pour le document DocBook **`testmml.xml`** en utilisant **`xmltex`** directement.

```
19: <xsl:template match="m:math">
20:  <xsl:copy>
21:   <xsl:apply-templates mode="math"/>
22:  </xsl:copy>
23: </xsl:template>
24:
25: <xsl:template  mode="math"
26:   match="*|@*|comment()|processing-instruction()|text()">
27:   <xsl:copy>
28:   <xsl:apply-templates    mode="math"
29:    select="*|@*|processing-instruction()|text()"/>
30:  </xsl:copy>
31: </xsl:template>
32:
33: <xsl:template match="programlisting" priority="4">
34:   <fo:block wrap-option="no-wrap"      whitespace-treatment="preserve"
35:             font-family="monospace"    font-size="9pt"
36:             space-before.optimum="4pt" space-after.optimum="4pt"
37:             text-align="start"
38:   >
39:     <xsl:apply-templates/>
40:   </fo:block>
41: </xsl:template>
42:
43: </xsl:stylesheet>
```

Les lignes 2, 4 et 5 definissent les préfixes pour les domaines nominaux : instructions XSLT (`xsl`), objets de formatage XSL (`fo`), éléments MathML (`m`), respectivement. À la ligne 7 nous importons la feuille de style `docbook.xsl` de Norman WALSH qui fait partie de sa distribution DocBook pour la transformation des éléments DocBook en objets de formatage. Cette commande `<xsl:import ...>` doit précéder les autres instructions XSLT si nous voulons que nos définitions remplacent celles éventuellement présentes pour des éléments identiques dans les feuilles de style standards importées. Les lignes 13 à 17 spécifient que les éléments `informalequation` et `inlineequation` doivent être copiés ainsi que leur contenu. D'une façon similaire (lignes 19 à 31) l'élément MathML `m:math` et tout l'arbre qu'il soutient doivent être copiés. Ce qui veut dire que les éléments MathML doivent être traités directement par l'application qui composera le source XSL-OF (dans notre cas PassiveTEX et `xmltex`). Finalement, les lignes 33 à 41 montrent notre définition pour les caractéristiques d'un élément `programlisting` utilisé pour un texte qui doit être représenté littéralement comme dans le fichier à l'entrée sans mise en forme supplémentaire (similaire à l'environnement `verbatim` de LATEX).

Après cette explication nous présentons ci-dessous les deux étapes nécessaires à l'obtention d'un document composé à partir du fichier `textmml.xml`.

```
xt testmml.xml foplus.xsl fotex.xml latex fotex.tex
```

Le résultat de ce traitement est montré à la figure 3. Il est évident que la mise en page est assez différente de celle de la figure 2. Ceci n'est pas une surprise parce que nous avons utilisé les feuilles de style XSLT standard de la distribution DocBook sans essayer de les personnaliser pour obtenir une mise en page similaire à celle de la classe `article` que nous avons utilisée dans le fichier `docbook.xmt` de la section 5.3.

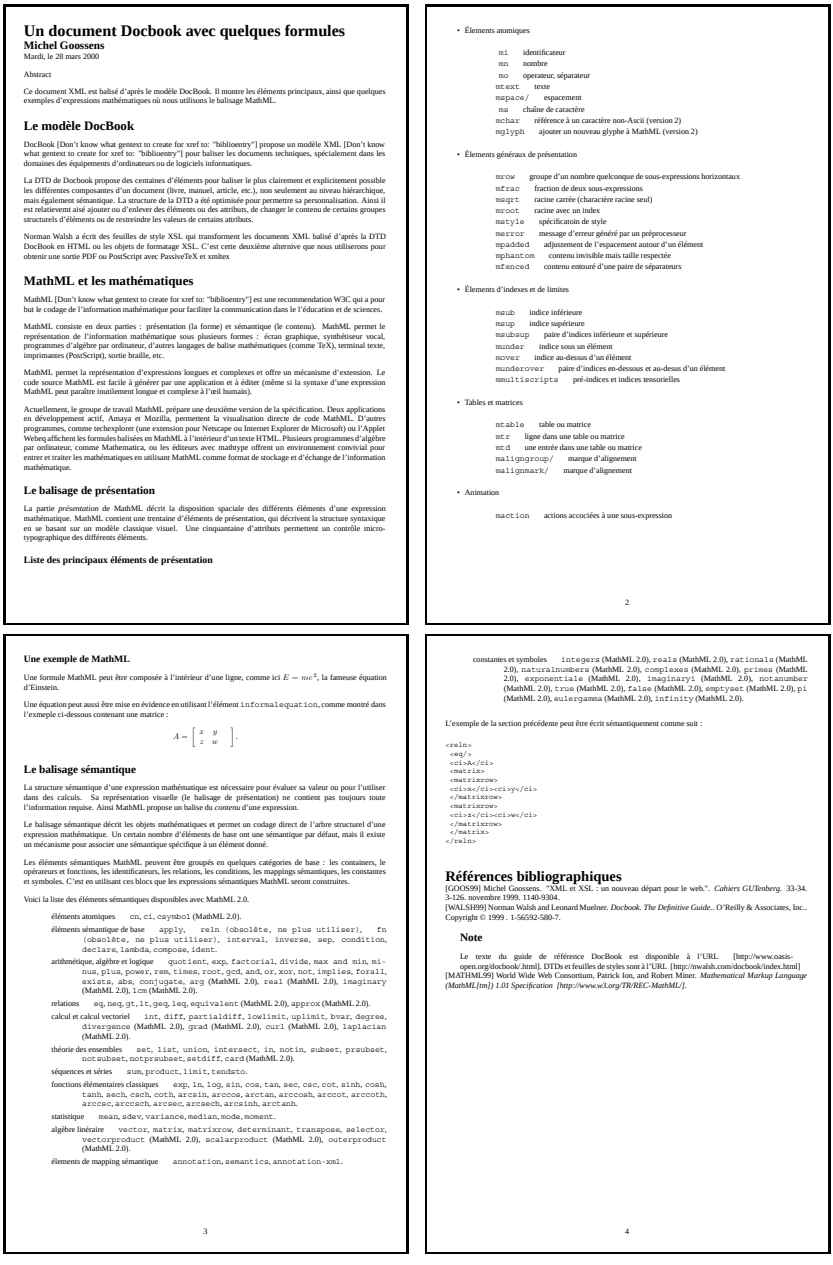
6. Conclusion : XML au cœur d'une stratégie pour la gestion du document électronique sur le web

Dans cet article nous nous sommes intéressés au traitement de fichiers XML par TEX en utilisant trois voies différentes et nous avons montré que ces approches sont applicables à plusieurs langages de balisage. En particulier nous avons étudié la composition de documents XML qui utilisent les DTD TEI et DocBook. Les formules mathématiques balisées en MathML peuvent également être traitées sans problèmes.

Pour la DTD TEI la figure 1 montre les résultats obtenus pour la composition du fichier `verlainetei.xml` à l'aide des trois procédures décrites aux sections 2 (A), 3 (B) et 4 (C), respectivement. Dans ce cas les trois approches sont en gros équivalentes en ce qui concerne l'aspect visuel de la sortie.

Pour la DTD DocBook et les mathématiques MathML nous avons également utilisé les méthodes (A) (figure 2) et (B) (figure 3). En principe nous aurions pu créer une feuille de style XSLT qui traduit le fichier source DocBook en instructions LATEX mais ceci ne nous aurait rien appris de nouveau parce que nous savons déjà (comparez les parties (A) et (C) de la figure 1) que nous pouvons définir nos commandes XSLT de façon à ce qu'elles correspondent littéralement aux définitions `xmltex` (voir la discussion à la section 4). L'apparence visuelle des approches (A) et (B) est assez différente mais dans les deux cas on peut facilement la personnaliser, si nécessaire.

L'approche générique (B), qui passe par l'intermédiaire du modèle abstrait des objets de formatage XSL-OF, nous semble plus ouverte parce qu'elle offre en principe la possibilité d'utiliser d'autres moteurs de composition. En effet, il est à espérer que d'autres outils pour la production de documents électronique, comme *Word* de Microsoft, *FrameMaker* d'Adobe et *WordPerfect* de Corel, pourront bientôt traiter les objets de formatage XSL-OF et les documents sources en XML. C'est ce que nous avons indiqué par la présence de l'ellipse verticale à la figure 4 qui montre la position centrale de XML dans la gestion et le traitement du document électronique.



Un document Docbook avec quelques formules

Michel Goossens

Mardi, le 28 mars 2000

Abstract

Ce document XML est balisé d'après le modèle DocBook. Il montre les éléments principaux, ainsi que quelques exemples d'expressions mathématiques où nous utilisons le balisage MathML.

Le modèle DocBook

DocBook [Don't know what gentext to create for xref to: "biblioentry"] propose un modèle XML [Don't know what gentext to create for xref to: "biblioentry"] pour baliser les documents techniques, spécialement dans les domaines des équipements d'ordinateurs ou de logiciels informatiques.

La DTD de Docbook propose des centaines d'éléments pour baliser le plus clairement et explicitement possible les différentes composantes d'un document (livre, manuel, article, etc.), non seulement au niveau hiérarchique, mais également sémantique. La structure de la DTD a été optimisée pour permettre sa personnalisation. Ainsi il est relativement aisé ajouter ou d'enlever des éléments ou des attributs, de changer le contenu de certains groupes structurels d'éléments ou de restreindre les valeurs de certains attributs.

Norman Walsh a écrit des feuilles de style XSL qui transforment les documents XML balisé d'après la DTD DocBook en HTML ou les objets de formatage XSL. C'est cette deuxième alternive que nous utiliserons pour obtenir une sortie PDF ou PostScript avec PassiveTeX et xmltex

MathML et les mathématiques

MathML [Don't know what gentext to create for xref to: "biblioentry"] est une recommendation W3C qui a pour but le codage de l'information mathématique pour faciliter la communication dans le l'éducation et de sciences.

MathML consiste en deux parties : présentation (la forme) et sémantique (le contenu). MathML permet le représentation de l'information mathématique sous plusieurs formes : écran graphique, synthétiseur vocal, programmes d'algèbre par ordinateur, d'autres langages de balise mathématiques (comme TeX), terminal texte, imprimantes (PostScript), sortie braille, etc.

MathML permet la représentation d'expressions longues et complexes et offre un mécanisme d'extension. Le code source MathML est facile à générer par une application et à éditer (même si la syntaxe d'une expression MathML peut paraître inutilement longue et complexe à l'œil humain).

Actuellement, le groupe de travail MathML prépare une deuxième version de la spécification. Deux applications en développement actif, Amaya et Mozilla, permettent la visualisation directe de code MathML. D'autres programmes, comme techexplorer (une extension pour Netscape ou Internet Explorer de Microsoft) ou l'Applet Webeq affichent les formules balisées en MathML à l'intérieur d'un texte HTML. Plusieurs programmes d'algèbre par ordinateur, comme Mathematica, ou les éditeurs avec mathtype offrent un environnement convivial pour entrer et traiter les mathématiques en utilisant MathML comme format de stockage et d'échange de l'information mathématique.

Le balisage de présentation

La partie *présentation* de MathML décrit la disposition spaciale des différents éléments d'une expression mathématique. MathML contient une trentaine d'éléments de présentation, qui décrivent la structure syntaxique en se basant sur un modèle classique visuel. Une cinquantaine d'attributs permettent un contrôle micro-typographique des différents éléments.

Liste des principaux éléments de présentation

- Éléments atomiques

  `mi` identificateur
  `mn` nombre
  `mo` operateur, séparateur
  `mtext` texte
  `mspace/` espacement
  `ms` chaîne de caractère
  `mchar` référence à un caractère non-Ascii (version 2)
  `mglyph` ajouter un nouveau glyphe à MathML (version 2)

- Éléments généraux de présentation

  `mrow` groupe d'un nombre quelconque de sous-expressions horizontaux
  `mfrac` fraction de deux sous-expressions
  `msqrt` racine carrée (characère racine seul)
  `mroot` racine avec un index
  `mstyle` spécificatoin de style
  `merror` message d'erreur généré par un préprocesseur
  `mpadded` adjustement de l'espacement autour d'un élément
  `mphantom` contenu invisible mais taille respectée
  `mfenced` contenu entouré d'une paire de séparateurs

- Éléments d'indexes et de limites

  `msub` indice inférieure
  `msup` indice supérieure
  `msubsup` paire d'indices inférieure et supérieure
  `munder` indice sous un élément
  `mover` indice au-dessus d'un élément
  `munderover` paire d'indices en-dessous et au-dessus d'un élément
  `mmultiscripts` pré-indices et indices tensorielles

- Tables et matrices

  `mtable` table ou matrice
  `mtr` ligne dans une table ou matrice
  `mtd` une entrée dans une table ou matrice
  `maligngroup/` marque d'alignement
  `malignmark/` marque d'alignement

- Animation

  `maction` actions accociées à une sous-expression

Une exemple de MathML

Une formule MathML peut être composée à l'intérieur d'une ligne, comme ici $E = mc^2$, la fameuse équation d'Einstein.

Une équation peut aussi être mise en évidence en utilisant l'élément `informalequation`, comme montré dans l'exemple ci-dessous contenant une matrice :

$$A = \begin{bmatrix} x & y \\ z & w \end{bmatrix}.$$

Le balisage sémantique

La structure sémantique d'une expression mathématique est nécessaire pour évaluer sa valeur ou pour l'utiliser dans des calculs. Sa représentation visuelle (le balisage de présentation) ne contient pas toujours toute l'information requise. Ainsi MathML propose un balise du contenu d'une expression.

Le balisage sémantique décrit les objets mathématiques et permet un codage direct de l'arbre structurel d'une expression mathématique. Un certain nombre d'éléments de base ont une sémantique par défaut, mais il existe un mécanisme pour associer une sémantique spécifique à un élément donné.

Les éléments sémantiques MathML peuvent être groupés en quelques catégories de base : les containers, le opérateurs et fonctions, les identificateurs, les relations, les conditions, les mappings sémantiques, les constantes et symboles. C'est en utilisant ces blocs que les expressions sémantiques MathML seront construites.

Voici la liste des éléments sémantiques disponibles avec MathML 2.0.

éléments atomiques `cn`, `ci`, `csymbol` (MathML 2.0).

éléments sémantique de base `apply`, `reln` (obsolète, ne plus utiliser), `fn` (obsolète, ne plus utiliser), `interval`, `inverse`, `sep`, `condition`, `declare`, `lambda`, `compose`, `ident`.

arithmétique, algèbre et logique `quotient`, `exp`, `factorial`, `divide`, `max` and `min`, `minus`, `plus`, `power`, `rem`, `times`, `root`, `gcd`, `and`, `or`, `xor`, `not`, `implies`, `forall`, `exists`, `abs`, `conjugate`, `arg` (MathML 2.0), `real` (MathML 2.0), `imaginary` (MathML 2.0), `lcm` (MathML 2.0).

relations `eq`, `neq`, `gt`, `lt`, `geq`, `leq`, `equivalent` (MathML 2.0), `approx` (MathML 2.0).

calcul et calcul vectoriel `int`, `diff`, `partialdiff`, `lowlimit`, `uplimit`, `bvar`, `degree`, `divergence` (MathML 2.0), `grad` (MathML 2.0), `curl` (MathML 2.0), `laplacian` (MathML 2.0).

théorie des ensembles `set`, `list`, `union`, `intersect`, `in`, `notin`, `subset`, `prsubset`, `notsubset`, `notprsubset`, `setdiff`, `card` (MathML 2.0).

séquences et séries `sum`, `product`, `limit`, `tendsto`.

fonctions élémentaires classiques `exp`, `ln`, `log`, `sin`, `cos`, `tan`, `sec`, `csc`, `cot`, `sinh`, `cosh`, `tanh`, `sech`, `csch`, `coth`, `arcsin`, `arccos`, `arctan`, `arccosh`, `arcsot`, `arccoth`, `arccsc`, `arccsch`, `arcsec`, `arcsech`, `arcsinh`, `arctanh`.

statistique `mean`, `sdev`, `variance`, `median`, `mode`, `moment`.

algèbre linéraire `vector`, `matrix`, `matrixrow`, `determinant`, `transpose`, `selector`, `vectorproduct` (MathML 2.0), `scalarproduct` (MathML 2.0), `outerproduct` (MathML 2.0).

éléments de mapping sémantique `annotation`, `semantics`, `annotation-xml`.

constantes et symboles `integers` (MathML 2.0), `reals` (MathML 2.0), `rationals` (MathML 2.0), `naturalnumbers` (MathML 2.0), `complexes` (MathML 2.0), `primes` (MathML 2.0), `exponentiale` (MathML 2.0), `imaginaryi` (MathML 2.0), `notanumber` (MathML 2.0), `true` (MathML 2.0), `false` (MathML 2.0), `emptyset` (MathML 2.0), `pi` (MathML 2.0), `eulergamma` (MathML 2.0), `infinity` (MathML 2.0).

L'exemple de la section précédente peut être écrit sémantiquement comme suit :

```
<reln>
 <eq/>
 <ci>A</ci>
 <matrix>
 <matrixrow>
 <ci>x</ci><ci>y</ci>
 </matrixrow>
 <matrixrow>
 <ci>z</ci><ci>w</ci>
 </matrixrow>
 </matrix>
</reln>
```

Références bibliographiques

[GOOS99] Michel Goossens. "XML et XSL : un nouveau départ pour le web.". *Cahiers GUTenberg*. 33-34. 3-126. novembre 1999. 1140-9304.
[WALSH99] Norman Walsh and Leonard Muelner. *Docbook. The Definitive Guide*.. O'Reilly & Associates, Inc.. Copyright © 1999 . 1-56592-580-7.

Note

Le texte du guide de référence DocBook est disponible à l'URL [http://www.oasis-open.org/docbook/.html]. DTDs et feuilles de styles sont à l'URL [http://nwalsh.com/docbook/index.html]
[MATHML99] World Wide Web Consortium, Patrick Ion, and Robert Miner. *Mathematical Markup Language (MathML[tm]) 1.01 Specifjcation [http://www.w3.org/TR/REC-MathML/].*

FIGURE 3: Sortie générée pour le document MathML `testmml.xml` en utilisant les feuilles de styles XSLT et PassiveTEX.

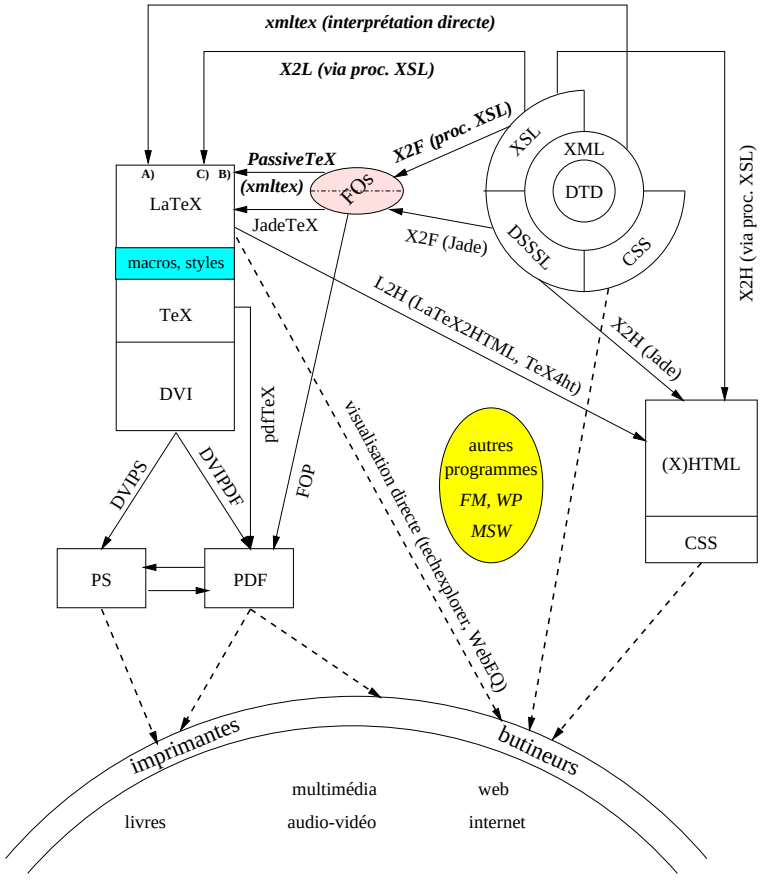


FIGURE 4: Stratégies XML pour le web

La figure 4 contient également dans sa partie supérieure nos trois approches A), B) et C) pour composer un document XML avec TEX. Elles sont identifiées chacune par un texte en italique et par une étiquette se trouvant au point d'arrivée de chaque flèche dans la boîte LATEX.

Plus généralement, la figure 4 montre comment XML peut être considéré comme élément central d'une stratégie du document électronique pour le web. Nous y avons en haut à droite représenté une classe de documents XML, caractérisés par une DTD. Ces documents peuvent être transformés en TEX par les procédures présentées dans cet article ou en HTML pour une visualisation avec des butineurs classiques en utilisant un feuille de style XSLT (et une feuille de style CSS pour mieux contrôler les divers aspects de la présenta-

tion). À l'inverse, on peut aussi générer une représentation XML (XHTML) à partir de fichiers source LaTeX à l'aide de programmes comme LaTeX2HTML ou TeX4ht. En traduisant les documents LaTeX en XML il devient plus facile de combiner des fragments de documents électroniques venant de plusieurs sources, un atout important pour le web.

Le jour n'est pas loin où les butineurs modernes et d'autres programmes multimédia pourront directement interpréter les sources XML. Il est plus que probable que des outils comme *MS Word*, *FrameMaker* et *WordPerfect* pourront également traiter les documents XML directement. Ainsi XML deviendra bientôt l'élément central dans une stratégie globale pour la gestion du document électronique en permettant le stockage, la conservation et l'utilisation de l'information par différentes applications et sur diverses plateformes informatiques.

Annexes

Les chapitres de cette annexe sont la traduction française de la première partie du manual `xmltex` [4].

A. Utiliser `xmltex`

A priori `xmltex` ne connaît rien concernant un type particulier de fichier XML et doit impérativement charger des fichiers externes supplémentaires pour avoir accès aux informations spécifiques nécessaires. Dans cette section nous décrivons comment l'information présente dans le fichier XML détermine quels fichiers seront chargés.

1. Si le fichier commence avec une marque d'ordre d'octet (MOO), le codage par défaut est UTF-16, sinon le décodage par défaut est UTF-8.
2. Si (après une MOO facultative) le document commence par une déclaration XML spécifiant un codage, ce codage sera utilisé, sinon le codage par défaut sera utilisé. On chargera un fichier ayant le nom *codage*`.xmt`. Ce fichier contient la correspondance entre le codage en question et les codes Unicode (ce fichier est obligatoire, s'il n'est pas trouvé une erreur TeX est générée).
3. Si le document a une déclaration `DOCTYPE` qui inclut un sous-ensemble interne, celui-ci sera analysé. S'il y a un appel d'entité à une DTD externe, les identificateurs de type `SYSTEM` ou `PUBLIC` de cette entité sont recherchés dans un catalogue. Si l'un ou l'autre identificateur est localisé dans le catalogue le module `xmltex` correspondant sera chargé.
4. Après le traitement du sous-ensemble interne, si la déclaration `DOCTYPE` spécifie une entité externe, les identificateurs de type `PUBLIC` ou `SYSTEM` du fichier DTD externe seront recherchés d'une façon similaire dans le catalogue et le fichier `xmltex` correspondant sera chargé s'il est connu.

5. Pendant le traitement par TEX un élément sera connu de `xmltex` en vertu d'un des modules chargés ou il sera inconnu. Si l'élément est inconnu mais fait partie d'un domaine nominal, l'URL de ce domaine nominal (pas le préfixe) est recherché dans le catalogue `xmltex`. Si le catalogue associe un module `xmltex` à ce domaine nominal le module en question est chargé. Par contre si l'élément inconnu ne fait pas partie d'un domaine nominal déclaré, le nom de l'élément lui-même sera recherché dans le catalogue.
6. Si après toutes ces étapes l'élément est toujours inconnu un message d'avertissement ou d'erreur sera généré (le comportement détaillé dépend des paramètres de configuration). La version actuelle de `xmltex` affiche seulement un message d'avertissement.

A.1. Les commandes de type catalogue

Comme discuté ci-dessus, `xmltex` a besoin d'un fichier TEX correspondant aux identificateurs de type `PUBLIC` ou `SYSTEM`, l'URL du domaine nominal, et les noms d'élément. Cette correspondance s'exprime à l'aide des commandes suivantes :

```
\NAMESPACE{URI}{fichier-xml}
\PUBLIC{FPI}{nom de fichier}
\SYSTEM{URI}{nom de fichier}
\NAME{nom d'élément}{fichier-xml}
\XMLNS{nom d'élément}{URI}
```

Si le premier argument d'une de ces commandes est égal à la chaîne de caractères spécifiée dans le fichier source XML, les commandes TEX présentes dans le fichier spécifié comme deuxième argument seront chargées. Les entrées de type `PUBLIC` et `SYSTEM` dans le catalogue peuvent être utilisées pour contrôler quel fichier XML sera chargé en réponse à un appel d'entité. D'un autre côté, la commande `\XMLNS` est toute différente ; si un élément dans le domaine nominal nul n'offre pas de définition pour l'élément en question, cette déclaration change le domaine nominal par défaut en le mettant égal à l'URI (*Uniform Resource Identifier* ou identificateur universel d'une ressource [12]) donné. Ensuite la consultation du catalogue est répétée, ce qui permet par exemple de contraindre des documents commençant avec l'élément `<html>` d'utiliser le domaine nominal de `xhtml`.

Ces commandes peuvent être placées dans un fichier de configuration. Si elles se trouvent dans le fichier `xmltex.cfg` elles s'appliquent à tous les documents. Lorsqu'elles se trouvent seulement dans un fichier de configuration de type « `\jobname.cfg` » (par exemple `verlainetei2000.cfg`) les commandes s'appliquent seulement au document indiqué.

A.2. Configurer `xmltex`

Les commandes de type « catalogue » décrites précédemment ne sont pas les seules qui peuvent être présentes dans les fichiers de configurations. Cette section présente une série d'autres commandes qui peuvent y être utilisées.

`\xmltraceonly`

Cette commande instruit le processeur de se limiter à l'analyse du document sans le composer. Les fichiers externes spécifiés dans le catalogue sont quand-même chargés, de sorte que l'analyse puisse signaler d'éventuels éléments pour lesquels aucun code n'est défini. En cas d'erreurs inconnues il est toujours intéressant d'utiliser `xmltex` dans ce mode pour isoler ce type de problèmes.

`\xmltraceoff`

Par défaut `xmltex` génère un journal avec les détails de l'analyse du fichier XML, indiquant chaque début et fin d'élément. La commande `\xmltraceoff` qui peut être utilisée dans les fichiers `xmltex.cfg` ou « `\jobname.cfg` » met fin à la génération de ces détails.

`\inputonce`{*fichier-xmt*}

Les entrées de type catalogue spécifient quels fichiers doivent être chargés si certaines constructions XML sont rencontrées. Alternativement la commande `\inputonce` chargera un fichier inconditionnellement, avec le système ignorant toute demande ultérieure de chargement. Ceci peut s'avérer particulièrement utile si on désire générer un format TEX incluant `xmltex`.

`\UnicodeCharacter`{*hex-ou-dec*}{*code-tex*}

Le premier argument est le code d'un caractère Unicode [17], au même format qu'un appel de caractère XML, à savoir un nombre décimal, ou un nombre hexadécimal majuscule précédé de la lettre petit « `x` ». Le deuxième argument est le code TEX qui sera utilisé pour composer ce caractère. N'importe quel code dans la gamme de validité XML (c.-à-d. de 0 à `x10FFFF`) peut être spécifié. Par exemple, on associe le code hexadécimal `0142` au caractère polonais `ł` (code TEX `\l`) par la commande :

```
\UnicodeCharacter{x0142}{{\l}}
```

Les codes XML dans la gamme 0-127 (caractères Ascii) peuvent également être spécifiés, mais les définitions fournies pour ces caractères ne seront pas utilisées par défaut. La définition en question sera toutefois sauvegardée pour être utilisée quand le caractère sera activée avec la commande `\ActivateASCII` décrite ci-dessous.

`\ActivateASCII`{*hex-ou-dec*}

L'argument de cette commande est un nombre inférieur à 128. Si un caractère est activé à l'aide de cette commande placée dans un fichier de configuration les instructions de composition spécifiques définies pour ce caractère seront exécutées chaque fois que le caractère apparaît à l'intérieur de données textuelles.

Certains caractères Ascii sont activés par défaut, en particulier tous ceux qui ont une signification spéciale pour TEX ou XML.

```
1: \ActivateASCII{x5C}
2: \UnicodeCharacter{x5C}{\textbackslash}
3: \ActivateASCII{123}
4: \ActivateASCII{125}
```

La ligne 1 active la barre oblique inversée et la ligne 2 la fait correspondre à la commande TEX `\textbackslash`. Les lignes 3 et 4 activent, respectivement, l'accolade gauche et droite (voir par exemple la table 10 de [1] ou [17]).

Si un format est généré, il y a essentiellement deux copies de `xmltex.cfg` qui peuvent jouer un rôle. Le fichier de configuration chargé au moment de la génération du format contrôlera des entrées catalogue et les extensions incluses dans le format. Un fichier `xmltex.cfg`, probablement différent du premier, pourra être chargé à chaque exécution pour permettre des commandes spécifiques.

Un fichier de configuration spécifique au document XML traité est toujours chargé juste après `xmltex.cfg`.

B. Les fichiers d'extension `xmltex`

Les fichiers d'extension `xmltex` sont le lien entre le fichier source balisé en XML et le code TEX de composition. Ils utilisent une syntaxe TEX (plutôt qu'XML) et ils peuvent charger directement ou indirectement d'autres fichiers, y compris des fichiers de classe ou d'extensions LATEX. Par exemple un fichier chargé pour un type particulier de document peut directement exécuter la commande `\LoadClass{article}`, ou alternativement il peut instruire un certain élément XML dans le document d'exécuter la commande `\documentclass{article}`. Dans les deux cas le document en question sera composé d'après le modèle typographique pré-défini de la classe `article.cls`. Il faut être attentif au fait que les fichiers d'extension peuvent être chargés à des moments particuliers, par exemple la première fois qu'un domaine nominal est déclaré dans un document, et ainsi le code doit pouvoir être exécuté à l'intérieur d'un groupe local.

À l'intérieur des fichiers d'extension `xmltex` les caractères ont leur sens LATEX habituel sauf que les fins de ligne sont ignorées de sorte qu'il n'est pas nécessaire d'ajouter un caractère `%` à la fin des lignes dans la définition des macros TEX. À la différence des conventions des fichiers de définition de police de type `fd`, les autres types d'espace blanc ne sont *pas* ignorés.

B.1. Liste des commandes

`\FileEncoding`{*codage*}

C'est l'analogue TEX de la déclaration de codage XML. En l'absence d'une déclaration explicite le fichier est supposé codé en UTF-8.

`\DeclareNamespace`{*préfixe*}{*URI*}

Cette commande déclare que *dans le fichier courant* le préfixe indiqué sera utilisé pour se rapporter aux éléments dans le domaine nominal spécifié par le URI. Un préfixe vide déclare le domaine nominal par défaut (sinon les noms d'élément sans préfixe sont considérés comme n'appartenant à aucun domaine nominal).

Notez que les éléments dans le document XML lui-même peuvent utiliser un préfixe différent, ou pas de préfixe du tout pour accéder à ce domaine nominal. Afin de distinguer ces différents préfixes pour un même domaine nominal, chaque fois qu'un domaine nominal est rencontré pour la première fois (dans une commande `\DeclareNamespace` dans un fichier chargé précédemment ou dans une déclaration de domaine nominal dans le fichier XML) un nouveau numéro lui est associé et toute déclaration de domaine nominal futur pour le même URI se limitera à associer un préfixe avec ce numéro. Ce sont ces numéros qui sont affichés dans le journal donnant les détails de l'analyse du document. Similairement, quand un élément quelconque est écrit dans un fichier externe il aura un préfixe normalisé contenant ce numéro indépendamment du préfixe qu'il avait initialement (les préfixes numériques étant illégaux en XML, nous nous assurons ainsi que ces formes internes ne peuvent être en désaccord avec un préfixe réellement utilisé dans le document source XML).

Trois domaines nominaux sont pré-définis : le domaine nominal nul (0), le domaine nominal XML (`http://www.w3.org/1998/xml`) (1) qui est pré-défini avec le préfixe `xml` en accord avec la recommandation W3C définissant les domaines nominaux [21], et le domaine nominal `xmltex` (`http://www.dcarlisle.demon.co.uk/xmltex`) (2) qui n'a pas de préfixe par défaut mais peut être utilisé pour permettre une syntaxe XML pour certaines commandes internes (par exemple pour des fichiers `aux` totalement en XML, actuellement ils mélangent d'une façon hybride les syntaxes TEX et XML).

`\XMLelement`{*nomq-élément*}{*spéc-attribut*} {*code-début*}{*code-fin*}

Cette commande est semblable à la commande LATEX `\newenvironment`.

Elle définit le code à exécuter au début et à la fin de chaque occurrence du type d'élément spécifié dans le premier argument. Ce code est exécuté à l'intérieur d'un groupe local (tout comme un environnement LATEX). Le deuxième argument déclare une liste d'attributs et leurs valeurs par défaut en utilisant la commande de `\XMLattribute` décrite ci-dessous.

`\XMLelement`{*nomq-élément*} {*spéc-attribute*} `{\xmlgrab}`{*code-fin*}

Une forme spéciale de la commande précédente. Elle exécute uniquement la commande `\xmlgrab` lorsque la balise de début de l'élément est rencontrée. Le code associé à la balise de fin *`code-fin`* a accès au contenu de l'élément (en syntaxe XML) comme `#1`. Son contenu n'est pas tout à fait identique au document initial, les domaines nominaux, les espaces blancs et les symboles de guillemet d'attribut ayant tous été normalisés.

`\XMLattribute`{*nomq-attribut*} {*nom-commande*}{*défaut*}

Cette commande peut seulement être utilisée comme deuxième argument d'une commande `\XMLelement`. Son premier argument indique le nom qualifié d'un attribut (utilisant un des préfixes de domaine nominal définis pour le fichier d'extension courant, qui peuvent être différents des préfixes utilisés dans le document). Son second argument spécifie la commande TEX à utiliser pour accéder à la valeur de cet attribut dans le code associé à la balise de début et de fin pour l'élément (l'utilisation de la syntaxe TEX à cet endroit permet un nom indépendant des déclarations de domaine

nominal qui sont courantes quand le code sera exécuté). Son troisième argument définit une valeur par défaut qui sera utilisée au cas où l'attribut n'est pas spécifié dans une instance de l'élément en question.

Si aucune valeur n'est spécifiée pour cet élément l'atome spécial `\inherit` mettra la valeur de la commande égale à celle définie dans un élément ancêtre.

En utilisant un atome TEX comme `\relax` comme valeur par défaut le code exécuté pour l'élément peut distinguer le cas où l'attribut n'est pas utilisé dans le document.

`\XMLnamespaceattribute` {*préfixe*}{*nomq-attribut*} {*nom-commande*}{*défaut*}

Cette commande est semblable à `\XMLattribute` mais elle est utilisée au niveau supérieur du fichier d'extension, pas dans l'argument d'une commande `\XMLelement`. Elle est équivalente à une spécification de l'attribut pour *chaque* élément dans le domaine nominal défini par le premier argument. Comme ailleurs le préfixe (une valeur nulle `{}` dénote le domaine nominal par défaut) se rapporte aux déclarations de domaine nominal dans les fichiers `xmltex` : les préfixes utilisés dans le document peuvent être différents.

`\XMLentity`{*nom*}{*code*}

Déclare une entité (interne analysable). Cette commande est équivalente à une déclaration `<!ENTITY`, sauf que le texte de remplacement est spécifié en syntaxe TEX.

`\XMLname`{*nom*}{*nom-commande*}

Déclare la commande TEX qui contiendra la forme interne et normalisée du nom XML spécifié dans le premier argument. Ceci permet au code spécifié dans une commande `\XMLelement` de se rapporter à des noms d'élément XML sans connaître les codages ni les préfixes de domaine nominal utilisés dans le document. Particulièrement utile dans ce contexte est la possibilité de comparer un nom à la valeur de `\XML@parent` ce qui permettra au code associé à un élément d'initier des actions différentes dépendant du nom de son parent.

`\XMLstring`{*nom-commande*}<>*données XML*</>

Cette commande sauvegarde le fragment de code XML donné au second argument dans la commande TEX spécifié dans le premier argument. Ceci peut être particulièrement utile pour redéfinir des chaînes de caractères invariables définies dans les classes de document LATEX pour réaliser certaines règles de composition spécifiées pour des caractères individuels.

Cette commande est également commode pour définir des chaînes de caractères qui sont utilisées dans les tests de comparaison avec des chaînes de caractères de documents XML. L'utilisation de la commande `\XMLstring` plutôt que `\def` garantit que les caractères et les codages dans la chaîne de caractères sont correctement normalisés.

C. Le traitement XML

`xmltex` s'efforce d'être un processeur XML non validateur entièrement conforme à la norme XML. Toutefois il reste quelques points à régler pour atteindre une conformité totale.

Peu d'erreurs sont signalées. La composition des noms n'est pas vérifiée par rapport à la liste des caractères permis, et diverses autres contraintes ne sont pas imposées.

Un processeur XML non validateur n'est pas obligé de lire des entités DTD externes (et `xmltex` ne le fait pas). Par contre il est obligé de lire le sous-ensemble interne et de traiter les définitions d'entité et les déclarations d'attributs. Le traitement des déclarations d'entités est assez satisfaisant : les entités paramètre sont traités comme décrit antérieurement et résultent dans le chargement d'un fichier d'extension `xmltex` correspondant ci celui-ci existe. Les entités externes sont traitées d'une façon similaire. Le résultat est le chargement d'un fichier XML, où à la différence du cas précédant, si le nom de l'entité n'est pas défini par une commande de type catalogue, `xmltex` utilisera directement l'identificateur `SYSTEM` pour charger (`\input`) le fichier en question, comme il s'agit souvent d'une ressource locale. Les entités analysables internes et entités paramètre sont essentiellement traitées comme des macros TEX, alors que les entités non analysables sont sauvegardées avec leur type `NDATA` pour éventuellement être utilisées par une commande comme `\includegraphics`.

Les valeurs par défaut pour les attributs sont traitées dans le sous-ensemble local de la DTD. Toutefois ces valeurs par défaut ne tiennent pas compte du domaine nominal et sont appliquées uniquement aux éléments ayant le même préfixe et le même nom local, à la différence du traitement des valeurs par défaut de la commande `\XMLattribute`.

Le support pour les codages dépend de la présence d'un fichier de mapping. Tout codage à huit bits dont les premières 127 positions coïncident avec celles d'Unicode peut être utilisé en créant un fichier de mapping simple. Par exemple, pour Koi8-r, un codage pour le cyrillique utilisé sur Unix en Russie le fichier de mapping en question (`koi8-r.xmt`) contient une série de déclarations du type :

```
\InputCharacter{x80}{x2500}
\InputCharacter{x81}{x2502}
```

où chaque ligne définit le mapping d'un code à l'entrée vers un code Unicode (p.ex. `x80` vers `x2500`).

Il y a un support complet pour le codage UTF-8, alors que pour le codage UTF-16 le support est minimal. Actuellement seules les valeurs dans la zone latin-1 fonctionnent ; dans cette zone UTF-16 est latin-1 avec un octet nul inséré avant (machine grand-boutienne) ou après (machine petit-boutienne) chaque octet. Le traitement de UTF-16 ignore cet octet nul puis traite le fichier comme s'il était en latin-1. En principe on pourrait utiliser un traitement similaire pour les autres codages 8 bits de la série latin en rendant les caractères Ascii actifs, mais ce traitement ne sera jamais vraiment satisfaisant avec TEX. Il est à espérer qu'une réalisation 16 bit de TEX tel qu'Omega proposera une solution adéquate à ce problème.

D. Communiquer directement avec TEX

En théorie spécifier des commandes `\XMLelement` (voir section B.1) devrait suffire pour piloter le traitement d'un document XML par `xmltex` Toutefois, parfois il est utile de pouvoir introduire des commandes TEX directement dans le fichier source pour mieux contrôler la composition et `xmltex` offre deux mécanismes dans ce domaine.

Le premier mécanisme est l'utilisation d'un domaine nominal propre à `xmltex`, qui correspond à un petit ensemble (actuellement vide) de constructions TEX auxquelles on accède par la syntaxe XML. Par exemple si `xmltex` proposait un mécanisme pour écrire des fichiers de type `toc` en syntaxe XML (plutôt que LATEX), il aurait besoin de l'équivalent de la commande `\contentsline` qui pourrait être un élément de la forme `<xmltex:contentsline>`... où le préfixe `xmltex` est déclaré sur l'élément en question ou un de ses ancêtres comme `xmlns:xmltex="http://www.dcarlisle.demon.co.uk/xmltex"`.

Actuellement le domaine nominal `xmltex` existe mais il est vide. Il est donc probablement plus utile de définir son propre domaine nominal pour introduire des commandes spécifiques TEX en chargeant un fichier d'extension qui contient les définitions TEX spécifiques à ce domaine nominal. Il faudra indiquer au minimum la correspondance entre le domaine nominal et le fichier d'extension à l'aide d'une commande `\NAMESPACE`. Par exemple si on trouve un élément XML `<clearpage xmlns="/mg/tex/spécial"/>` dans un document XML (génération d'un saut de page) on devra avoir quelque part la commande `\NAMESPACE{/mg/tex/spécial}{mgspec.xmt}`, où le fichier `mgspec.xmt` contient les lignes :

```
\DeclareNamespace{mgspec}{/mg/tex/spécial}
\XMLelement{mgspec:clearpage}{\clearpage }
```

Le deuxième mécanisme utilise une instruction de traitement XML. Elle permet de spécifier une ou plusieurs commandes TEX qui seront exécutées à cet endroit. Un exemple d'une instruction de traitement XML qui génère une nouvelle ligne est : `<?xmltex \newline ?>`.

Un autre exemple est quand quelque chose tourne mal et on obtient l'invite `*`. On peut alors interrompre l'exécution de `xmltex` à l'aide de l'instruction de traitement : `<?xmltex \stop?>`.

Références bibliographiques

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

[2] Apache XML Project. *FOP, XSL Formatting Object Processor in Java.* `http://xml.apache.org/fop/`

[3] Lou Burnard et C.M. Sperberg-McQueen. « La TEI simplifiée : une introduction au codage des textes électroniques en vue de leur échange. » *Cahiers GUTenberg*, 24, pages 23–152, juin 1996.

[4] David Carlisle. *xmltex A non validating (and not 100% conforming) namespace aware XML parser implemented in TEX*. Se trouve sur les sites CTAN dans le répertoire `macros/xmltex/`.

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

[6] Daniel Glazman. *CSS2. Feuilles de style HTML.* Éditions Eyrolles, Paris, 1999.

[7] Michel Goossens. « XML et XSL : un nouveau départ pour le web. » *Cahiers GUTenberg*, 33-34, pages 3–126, novembre 1999.

[8] Michel Goossens et Sebastian Rahtz. *The LATEX Graphics Companion.* Addison-Wesley, Reading, 1997.

[9] Michel Goossens et Sebastian Rahtz. *The LATEX Web Companion.* Addison-Wesley, Reading, 1999.

[10] Yannis Haralambous et John Plaice. « , une extension de \TeX{} incluant Unicode et des filtres de type Lex. » *Cahiers GUTenberg*, 20, pages 55–80, mai 1995.

Voir également l'URL `http://www.gutenberg.eu.org/omega/`

[11] John Hobby. *A user's manual for MetaPost.* Computer Science Technical Report 162, AT&T Bell Laboratories, 1992.

MetaPost et son manuel font partie de toutes les distributions TEX. Des contributions de macros MetaPost sont disponibles sur CTAN dans le répertoire `graphics/metapost/`. Metapost est également décrit au chapitre 3 dans [8].

[12] 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`

[13] International Organization for Standardization. *Information Technology—Processing Languages—Document Style Semantics and Specification Language (DSSSL). First edition, 1996* International Standard ISO/IEC 10179:1996.

Une version PDF à usage personnel est disponible sur l'Internet à l'URL `ftp://ftp.ornl.gov/pub/sgml/WG8/DSSSL/dsssl96b.pdf`.

[14] Alain Michard. *XML Langage et applications.* Éditions Eyrolles, Paris, 1999.

[15] Sebastian Rahtz. *Passive TEX*
http://users.ox.ac.uk/~rahtz/passivetex/

[16] Sebastian Rahtz. *TEI and XSL.* http://users.ox.ac.uk/~rahtz/tei/

[17] The Unicode Consortium. *The Unicode Standard, Version 3.0.* Addison-Wesley, Reading, 2000.

[18] Norman Walsh et Leonard Muelner. *Docbook. The Definitive Guide.* O'Reilly & Associates, Inc., Sebastopol, USA, 1999.
http://nwalsh.com/docbook/index.html

Le texte du guide de référence DocBook est disponible à l'URL http://www.oasis-open.org/docbook/.html. DTD et feuilles de styles sont à l'URL http://nwalsh.com/docbook/index.html

[19] 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

[20] World Wide Web Consortium. Patrick Ion et Robert Miner (rédacteurs). *Mathematical Markup Language (MathML[tm]) 1.01 Specification.* http://www.w3.org/TR/REC-MathML/

[21] World Wide Web Consortium. Tim Bray, Dave Hollander et Andrew Layman (rédacteurs). *Namespaces in XML.*
http://www.w3.org/TR/REC-xml-names

[22] World Wide Web Consortium, Jon Ferraiolo (rédacteur). *Scalable Vector Graphics (SVG) 1.0 Specification (W3C Working Draft).*
http://www.w3.org/TR/SVG

[23] 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

La traduction française de la spécification XML se trouve dans les *Cahiers GUTenberg*, 33-34, pages 191–280, novembre 1999 ainsi qu'à l'URL http://babel.alis.com/web_ml/xml/REC-xml.fr.html.

[24] World Wide Web Consortium, JamesClark (rédacteur). *XSL Transformations (XSLT), Version 1.0 (W3C Recommendation 16 November 1999).* http://www.w3.org/TR/xslt

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