Cahiers GUTenberg

UNE SOLUTION DE CONVERSION RTF VERS XML/MATHML AVEC PUBLICATION WEB DYNAMIQUE EN XML/MATHML

André Violante

Cahiers GUTenberg, nº 39-40 (2001), p. 181-200.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_2001___39-40_181_0>

Une solution de conversion RTF vers XML/MATHML avec publication Web dynamique en XML/MATHML

André VIOLANTE

Publilog `http://www.publilog.fr`

Résumé. Il existe divers moyens permettant de générer, en HTML, des pages Web contenant des formules mathématiques. Ces techniques passent le plus souvent par une phase de transformation statique des équations mathématiques en graphiques, voire en appliquettes ou en données de *plug-in*. Ceci ne facilite pas la mise à jour des sites et oblige souvent à l'utilisation d'environnements de navigation adaptés.

Une approche radicalement différente consiste à publier, de façon dynamique, directement en XML/MATHML. Pour cela, il est nécessaire de s'appuyer sur des techniques de traitement dites « côté serveur » (*server side*).

Dans cette présentation, nous exposerons une solution permettant de produire simplement des fichiers XML/MATHML à partir de fichiers RTF et de rapidement les publier sur Internet (ou autre) sans contrainte au niveau du client grâce à Cocoon, XSLT et TEX.

Mots-clés : RTF4XML, XML, MATHML, Cocoon, XSLT, TEX.

Abstract. *There exist several ways generating* HTML WEB *pages with mathematical material inside. For that purpose, we almost allways need to statically translate mathematical equations into images, applets or even plugging datas. Though, updating the Web site becomes a heavy task while clients will often need special navigation environments.*

A rather different approch is to dynamically publish, directly with XML/MATHML*. It's possible using "server side" technology.*

Within this presentation, we will expose a solution allowing a simple XML/MATHML *generation from* RTF *files and a fast publication over the Internet (or other) without constraints on the client side. This can be achieved with Cocoon,* XSLT *and TEX.*

Keywords: RTF4XML, XML, MATHML, *Cocoon*, XSLT, *TEX.*

1. Le choix initial de XML

Notre besoin initial était de convertir des fichiers RTF en LaTeX afin de composer des ouvrages pour des maisons d'édition. Mais, s'il est très bien de convertir, encore faut-il avoir confiance dans ce que l'on obtient. Il est difficile et inquiétant pour un rédacteur de recevoir une épreuve contenant des erreurs (ou des défauts) que ne contenait pas le fichier initial.

Il aurait été possible d'utiliser un des convertisseurs RTF/LaTeX existant. Cependant, les résultats obtenus nécessitaient des « scriptages » de transformation et de nettoyage importants. LaTeX a une syntaxe se prêtant mal à ce style de manipulations. Nous avons donc décidé de nous pencher plutôt sur une conversion LaTeX *via* XML car ce dernier se prêtait parfaitement aux manipulations automatiques que nous souhaitions effectuer.

D'autre part, les éditeurs avec lesquels nous travaillons, ont aujourd'hui pris conscience qu'un document, fabriqué sur des supports traditionnels et vendu dans des circuits classiques peut, demain, suivre des voies différentes. XML représente un moyen d'assurer la pérénité de leur fond documentaire.

Nous avons basé aujourd'hui notre système de composition sur un convertisseur RTF vers XML puis LaTeX.

2. Conversion en XML/MATHML de fichiers RTF

Il existe plusieurs moyens de convertir des fichiers RTF en XML :

- par des logiciels important du RTF et exportant du XML (FrameMaker, XMetal, EPic…) ;
- en utilisant des outils comme Majix, Filtrix, RTF2XML…

Aucune de ces méthodes n'est parfaite (mais qui peut se vanter de l'être). Mais leur plus gros défaut est d'ignorer la notation mathématique MATHML. Il se trouve que nous ne fabriquons que des ouvrages scientifiques. Partant de l'adage prétendant que l'on n'est jamais mieux servi que par soi-même, nous avons décidé de développer notre propre convertisseur.

2.1. Les difficultés

RTF est un format très complet, fini et relativement facile à parser. Les problèmes majeurs que l'on peut rencontrer sont liés :

- au grand nombre de commandes et à la gestion des cascades ;

– aux différences de styles de codage et de versions de format en fonction des logiciels qui en génèrent ;
– au laxisme et à l'anarchie structurels dont il peut faire preuve dans l'expression de certaines constructions. Il est, structurellement, l'antithèse de XML ;
– à la diversité des objets qu'il peut encapsuler (objets OLE divers, formats graphiques variés...).

Quoi qu'il en soit, cela est loin de présenter les mêmes difficultés que la conversion partant de TEX ou LATEX.

2.2. Les différents choix possibles

2.2.1. L'environnement de conversion

Tout d'abord, le choix de la plate-forme de conversion sera déterminant. Le format RTF provenant souvent de logiciels propriétaires, il est normal qu'il incorpore des objets dans des formats également propriétaires. Typiquement les formats graphiques WMF, EMF, BMP, etc., seront plus naturellement exploitables sous environnement Windows que sous Linux ou MacOS. Inversement, le format PICT ne peut être correctement converti que sous environnement MacOS. D'autre part, la technologie OLE (incorporation d'objets), est native dans Windows alors qu'elle n'existe pas sous cette forme sous Linux. Il était donc important de trouver le plus grand dénominateur commun. Windows nous a donc semblé être le meilleur candidat.

2.2.2. Les outils de développement

Le choix est vaste :

– Java, C/C++ ;
– Perl, Ominmark, Python ;
– ...

Tous présentent une série d'avantages propres. La performance (en terme de vitesse) n'étant pas un facteur déterminant, chacun d'eux pouvait convenir. Nous avons opté pour C++ car nous étions conscients qu'il faudrait s'appuyer parfois sur des fonctions internes du système. Ce langage permet de travailler à un niveau suffisamment bas. Il est orienté objet, autorise la définition de structures de données bien adaptées et génère, de plus, les exécutables les plus rapides.

2.2.3. *Le niveau de complétude et style de conversion*

Complétude. À ce niveau, il va de soi que l'idéal est d'être exhaustif. L'ampleur du travail est alors conséquente. Il faut cependant considérer qu'aucun logiciel n'exploite la totalité des commandes du format RTF. Par exemple, il est intrinsèquement possible en RTF de définir un format nommé de section (au même titre qu'un format de style paragraphe ou caractère). Cette fonction bien utile permettrait, par exemple, d'émuler le fonctionnement des environnements de LaTeX. Word n'exploite malheureusement pas cette caractéristique et je ne connais aucun logiciel de bureautique le faisant. On peut ainsi trouver une multitude de commandes jamais (ou très rarement) utilisées.

Nous nous sommes donc attachés à convertir prioritairement les éléments du langage générés suite à une manipulation de l'interface homme/machine des logiciels les plus courants.

Style. Le format RTF dispose de fonctionalités de travail sur feuille de style (paragraphe et caractère). Bien que servant à associer une présentation à certains contextes d'éléments, elles peuvent malgré tout être considérées comme des éléments structurants. Cela permet d'imaginer une conversion guidée par la philosophie que l'on trouve dans HTML avec CSS2 ou XML avec XSL c'est-à-dire visant à la dissociation maximale du contenu et de la présentation mais contenant intrinsèquement les deux.

La conversion doit générer un style de XML dans le même esprit que le couple XML/XSL mais avec une expression quelque peu différente. En effet, l'objectif est de restituer le maximum des informations contenues dans le fichier de départ mais également de permettre des transformations très variées (retrouver la présentation d'origine, filtrer certaines constructions, construire une autre structure…). Nous qualifions le fichier généré de « XML *flat* ». Même si cela est possible, nous ne voyons pas, dans l'immédiat, la nécessité de créer une DTD pour ces fichiers. Ils ne sont que des pivots. Le besoin d'une DTD se ferait plutôt sentir dans une phase aval de post-transformation.

2.3. RTF4XML

Ce programme convertit des fichiers au format RTF (jusqu'à la version 1.6) en XML. Il fonctionne sous Windows NT4/2000.

Il prend, en entrée, un fichier RTF et génère deux fichiers XML :

- un pour l'expression du contenu,
- l'autre pour l'expression de la feuille de style modélisant la feuille de style de présentation RTF.

2.3.1. Les formules mathématiques

Au niveau des formules, il convertit en MATHML (de présentation) les éléments « Éditeur d'équations » (version 1 à 3), « MathType » (version 3 à 4) et « Champs d'équations ». Il faut cependant noter, concernant ces derniers, que ce n'est pas exactement du MATHML qui est généré mais quelque chose qui s'en rapproche. Ceci est lié à l'anarchie qui peut régner dans l'expression de ce type de champs (imbrication de leur syntaxe propre avec celle du RTF). Il serait possible, par un post-traitement, de normaliser cela. Mais les champs d'équations sont aujourd'hui un moyen obsolète de saisir des mathématiques avec Word. Nous ne pensons donc pas travailler plus avant sur ce point.

2.3.2. Les graphiques

Les graphiques rencontrés au format WMF, EMF, PNG, TIFF, BMP, OLE sont convertis en EPS et en PNG . Si le schéma est placé par un champ d'inclusion de figure (figure externe) une référence au fichier est insérée. Le format PICT n'est pour l'instant pas pris en charge (mais la référence à un placement de figure est malgré tout générée).

2.3.3. Les autres éléments

Sont convertis :

- les formats caractères et paragraphes,
- les commutations de polices et de corps,
- les attributs paragraphes principaux,
- les attributs caractères principaux,
- les tables,
- les caractères spéciaux,
- les en-têtes et pieds de page,
- les notes de bas de page,
- les multicolonnes,
- les champs hyperlien, page, signet, reférence, index, équation,
- les cadres texte et graphique avec leurs attributs.

Il manque principalement :

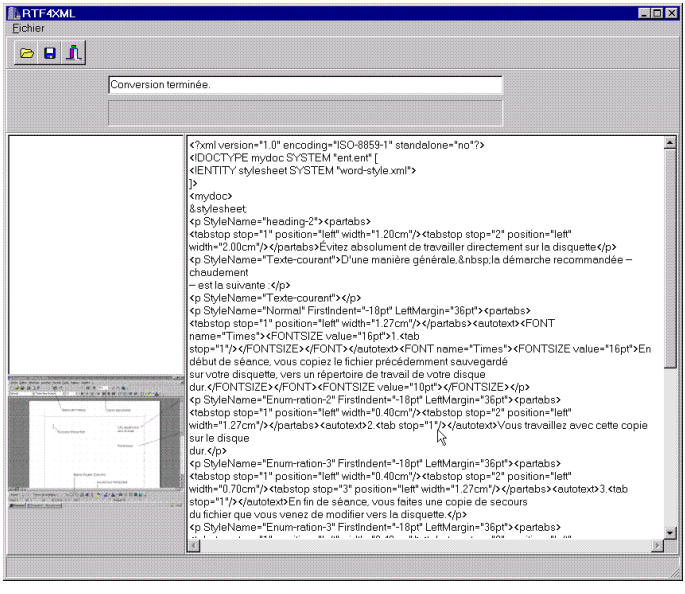
- les effets WordArt,
- l'interprétation complète du langage graphique,
- les tables des matières (est-ce nécessaire avec XML ?),
- les détourages (*clippings*) sur les insersions graphiques.

Ces listes ne sont certainement pas exhaustives. La tâche la plus ardue restant certainement la conversion généralisée du langage graphique. Pour cela, il serait intéressant de s'appuyer sur SVG.

2.4. La conversion

Elle s'effectue depuis une interface graphique (ce qui n'est pas réellement indispensable) ou en ligne de commande.

À la fin de l'exécution, une analyse est effectuée afin de s'assurer de la conformité du XML généré et qu'aucune définition d'entité ne manque. Si c'était le cas, il est possible de compléter manuellement le catalogue d'entités et de reconvertir. Par contre, si un message prévient d'une éventuelle non conformité, c'est que l'on est confronté à une erreur de conversion. Ce type d'erreur peut provenir d'une commande RTF non prise en charge mais incorrectement ignorée.



Exécution de RTF4XML

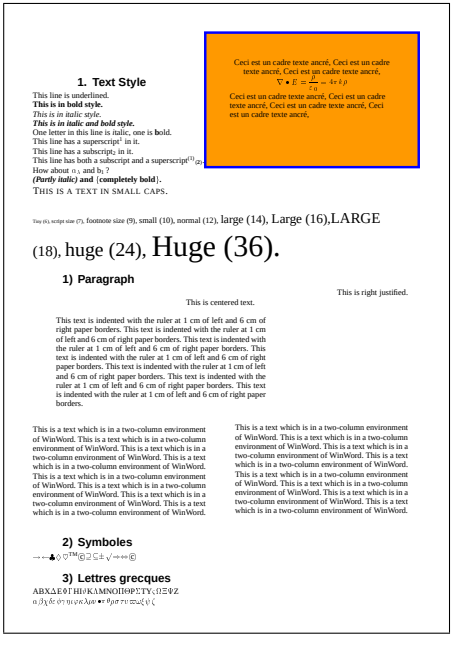
Une fois la conversion effectuée, nous arrivons à une phase capitale : la transformation. C'est l'essence de la polyvalence de XML.

2.5. Un premier type de transformation vers TEX

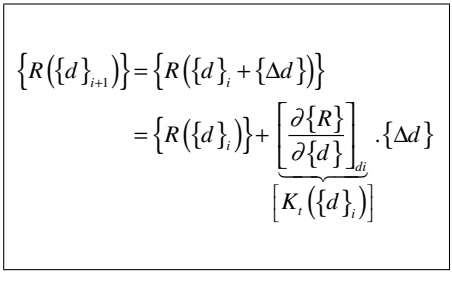
Un moyen simple et visuel permettant de constater que la conversion a correctement fonctionné consiste à transformer le fichier XML en LATEX en cherchant à reproduire l'apparence du document originel. Dans ce mode, on s'attache à émuler avec LATEX, dans la mesure du possible, les styles de présentation RTF.

Voici quelques exemples de ce type de traduction.

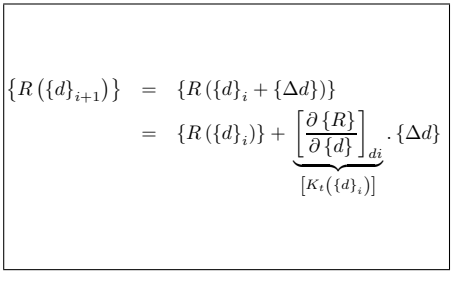
Fichier RTF



Restitution par TEX du fichier XML généré



Une équation MathType 3.5



La même généré par TEX à partir de MATHML

Voici, pour information, le fichier XML/MATHML généré ayant permi d'obtenir ce dernier exemple (c'est petit, mais il est vrai que MATHML est verbeux !) :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE mydoc SYSTEM "ent.ent" [
<!ENTITY stylesheet SYSTEM "eqw-style.xml">
]>
<mydoc>
&stylesheet;
<p StyleName='Normal'><math>
    <semantics>
        <mtable align='center' columnalign='right center left'>
            <mtr>
                <mtd>
                    <mrow>
                        <mfenced open='{' close='}'>
                            <mrow>
                                <mi>R</mi>
                                <mfenced open='(' close=')'>
                                    <mrow>
                                        <msub>
                                            <mfenced open='{' close='}'>
                                                <mrow>
                                                    <mi>d</mi>
                                                </mrow>
                                            </mfenced>
                                            <mrow>
                                                <mi>i</mi>
                                                <mo>+</mo>
                                                <mn>1</mn>
                                            </mrow>
                                        </msub>
                                    </mrow>
                                </mfenced>
                            </mrow>
                        </mfenced>
                    </mrow>
                </mtd>
                <mtd>
                    <mo>=</mo>
                </mtd>
                <mtd>
                    <mfenced open='{' close='}'>
                        <mrow>
                            <mi>R</mi>
                            <mfenced open='(' close=')'>
                                <mrow>
                                    <msub>
                                        <mfenced open='{' close='}'>
                                            <mrow>
                                                <mi>d</mi>
                                            </mrow>
                                        </mfenced>
                                        <mrow>
                                            <mi>i</mi>
                                        </mrow>
                                    </msub>
                                    <mo>+</mo>
                                    <mfenced open='{' close='}'>
                                        <mrow>
                                            <mi>&Delta;</mi>
                                            <mi>d</mi>
                                        </mrow>
                                    </mfenced>
                                </mrow>
```

```
                </mfenced>
            </mrow>
        </mfenced>
    </mtd>
</mtr>
<mtr>
    <mtd>
    </mtd>
    <mtd>
        <mo>=</mo>
    </mtd>
    <mtd>
        <mfenced open='{' close='}'>
            <mrow>
                <mi>R</mi>
                <mfenced open='(' close=')'>
                    <mrow>
                        <msub>
                            <mfenced open='{' close='}'>
                                <mrow>
                                    <mi>d</mi>
                                </mrow>
                            </mfenced>
                            <mrow>
                                <mi>i</mi>
                            </mrow>
                        </msub>
                    </mrow>
                </mfenced>
            </mrow>
        </mfenced>
        <mo>+</mo>
        <munder>
            <munder>
                <mrow>
                    <msub>
                        <mfenced open='[' close=']'>
                            <mrow>
                                <mfrac>
                                    <mrow>
                                        <mrow>
                                            <mo>&part;</mo>
                                            <mfenced open='{' close='}'>
                                                <mrow>
                                                    <mi>R</mi>
                                                </mrow>
                                            </mfenced>
                                        </mrow>
                                    </mrow>
                                    <mrow>
                                        <mrow>
                                            <mo>&part;</mo>
                                            <mfenced open='{' close='}'>
                                                <mrow>
                                                    <mi>d</mi>
                                                </mrow>
                                            </mfenced>
                                        </mrow>
                                    </mrow>
                                </mfrac>
                            </mrow>
                        </mfenced>
                        <mrow>
                            <mi>d</mi>
                            <mi>i</mi>
                        </mrow>
```

```
                        </msub>
                    </mrow>
                    <mo stretchy='true'>&undcub;</mo>
                </munder>
                <mrow>
                    <mfenced open='[' close=']'>
                        <mrow>
                            <msub>
                                <mi>K</mi>
                                <mrow>
                                    <mi>t</mi>
                                </mrow>
                            </msub>
                            <mfenced open='(' close=')'>
                                <mrow>
                                    <msub>
                                        <mfenced open='{' close='}'>
                                            <mrow>
                                                <mi>d</mi>
                                            </mrow>
                                        </mfenced>
                                        <mrow>
                                            <mi>i</mi>
                                        </mrow>
                                    </msub>
                                </mrow>
                            </mfenced>
                        </mrow>
                    </mfenced>
                </mrow>
            </munder>
            <mo>.</mo>
            <mfenced open='{' close='}'>
                <mrow>
                    <mi>&Delta;</mi>
                    <mi>d</mi>
                </mrow>
            </mfenced>
        </mtd>
    </mtr>
</mtable>
</semantics>
</math></p>
</mydoc>
```

2.6. Bon ! et alors ?

Dans l'état, il s'agit d'une exploitation directe du XML *flat* obtenu. Si l'on devait s'arrêter là, on pourrait évidemment s'interroger sur l'utilité de ce qui vient d'être fait. Pourquoi « s'amuser » à restituer une présentation RTF avec LaTeX en passant par XML ? Pour se servir de XML comme format d'archivage ? Pourquoi ne pas stocker des fichiers PDF issus d'une impression du RTF ?… ce serait plus simple (et plus fiable).

Le code LaTeX obtenu dans les exemples précédents, même s'il est relativement propre et structuré (en héritage de XML), serait difficilement exploitable car il ne vise qu'à une chose : restituer, autant que faire se peut, la présentation d'origine.

Nous procédons donc ensuite à un autre type de transformation visant plutôt à générer un document LaTeX très propre, lisible et bien structuré, utilisant par avance la syntaxe de la feuille de style avec laquelle nous devrons réaliser l'ouvrage (faisant appel à des `\chapter`, `\section`, `\subsection`, des environnements, etc.).

Voilà qui est mieux. On est maintenant en mesure de produire, dans de bonnes conditions, les ouvrages que l'on nous a confiés.

Mais les enjeux de XML/MATHML ne s'arrêtent pas là.

3. XML et la publication dynamique

Nous disposons dès lors d'un fichier XML. Nous avons vu qu'il était possible de le transformer en TeX de présentation (restitution approchée de la présentation RTF) ou en TeX de structure (plus adapté à notre travail de mise en forme).

Il est aussi possible de transformer simplement ce fichier en HTML. Concernant les formules mathématiques, plusieurs solutions s'offrent à nous :

- en faire des images à une résolution écran (laquelle ?) ;
- laisser du MATHML pour les navigateurs compatibles MATHML (Amaya, Mozilla,...), en prenant garde d'utiliser des noms valides pour les entités mathématiques ;
- consituer des données pour des *plug-ins* de navigateur (MATHML compatible WebEQ, TeX compatible TeXExplorer, ...).

Les fichiers, quelle que soit la voie choisie, devront être transférés sur un serveur WEB. Concernant l'ergonomie de navigation, elle pourra, de façon limitée, être spécifiée dans le fichier d'origine ou alors passera par une phase d'édition des fichiers HTML. Dans cette dernière hypothèse, toute reconversion nécessitera la réédition des fichiers HTML regénérés. Il faut, de plus, prévoir un fichier PDF qui permette de proposer une impression de qualité.

On le voit, les inconvénients de cette méthode sont nombreux :

- un grand nombre de fichiers sont nécessaires si l'on veut être indépendant des navigateurs ;
- difficulté de mise à jour ;
- difficulté d'administration.

Il est temps de vérifier que XML et XSL constituent bien, comme le disait Michel Goossens dans le numéro 33-34 (novembre 1999) des *cahiers GUTenberg* « un nouveau départ pour le WEB »

Dans ce document, il disait :

> « Dans l'environnement informatisé actuel, il est important que nos documents soient mis à la disposition d'une communauté aussi large que possible. Pour optimiser les possibilités de réutilisation de ces documents par les différents supports, visualisateurs, bases de données, etc., il est primordial que les documents soient clairement balisés structurellement, portables et indexables pour faciliter les recherches. »

XML permet effectivement de répondre à ces contraintes. Il serait donc intéressant de pouvoir publier directement du XML/MATHML sans passer par une phase de pré-conversion statique. Dès lors, nous avons l'alternative suivante.

Attendre la disponibilité de navigateurs adéquats. Il ne vont pas tarder. Des outils comme Mozilla ou Amaya ne sont pas loin de proposer des solutions réalistes. Mais il en va de la navigation comme du reste. Doit-on tendre vers l'uniformité ou accepter les différences ? Peut-on dire à un utilisateur : « si vous voulez voir nos pages, vos devez avoir le navigateur X ou Y » ? Même si un navigateur idéal et universel apparaissait, serait-il raisonnable de tenter de l'imposer de façon généralisée ?

S'affranchir du navigateur. Pour cela une solution, fréquemment utilisée, consiste à faire appel à des traitements sélectifs au niveau du navigateur client. Ceci est réalisé par du code Javascript intégré dans les pages servies. Cela oblige à placer, sur le serveur, des fichiers ne faisant plus du tout la séparation du contenu, de la présentation ni de la logique. Cette solution ne fait que contribuer au désordre existant.
Une méthode plus rationnelle consisterait à confier cette tâche aux serveurs. Pour cela, il faudrait qu'ils aient un comportement du type :

- si le navigateur client est capable de les interpréter, servir « verbatim » les données XML dont ils disposent ;
- sinon, se charger de la mise en compatibilité et faire intervenir dynamiquement les transformations nécessaires.

Ceci permet de garder une source d'information unique, générique et parfaitement propre. La génération dynamique (et aussi temporaire) des données compatibles avec le client n'ayant lieu que si cela est nécessaire. Cette façon de procéder, requiert l'utilisation de techniques de traitement dites « côté serveur » (*server side*).

3.1. Le traitement « côté serveur »

Celui-ci consiste à décharger les postes clients (les navigateurs) des tâches compliquées et trop souvent dépendantes des plate-formes et des environnements pour les confier au serveur WEB. De cette façon, nous serons sûr que

tous les clients seront servis de la même façon. Cela permettrait par exemple de servir au poste client des équations (en bitmap) adaptées à la résolution de l'écran depuis lequel il consulte la page, de lui permettre de les zoomer s'il le souhaitait et éventuellement, un jour, de consulter cette même page depuis un téléphone WAP.

À partir d'une seule instance XML et d'un fichier de transformation XSLT, le serveur peut prendre dynamiquement en charge :

- la conversion du document global XML en HTML et son service au poste client ;
- la conversion des parties MATHML en TEX puis en images, et leur placement ;
- la conversion des formats d'images des illustrations et leur placement ;
- la création automatique d'objets de navigation (liens) ;
- la création d'un fichier PDF en cas de demande d'impression.

Le tout en s'adaptant aux caractéristiques du poste du client. Si une de ses caractéristiques était de reconnaître, en natif, le XML/MATHML alors l'étape XML→ LATEX→GIF ne serait pas exécutée. Le serveur procéderait simplement à la recopie « verabtim » des données mathématiques qu'il détient.

Dans ce schéma, pour un même type de document et de présentation, un seul fichier de transformation XSLT est nécessaire pour servir un nombre quelconque de fichiers XML. Une fois le premier fichier mis en place, la fourniture de nouveaux contenus XML sera instantanée.

3.1.1. Les outils nécessaires

Le serveur. Un serveur Apache au-dessus duquel on a greffé Tomcat. Tomcat est un moteur « servlet JSP » permettant d'effectuer des traitements côté serveur s'appuyant sur Java.

Le contexte de publication. Au dessus de Tomcat se greffe Cocoon, un « contexte de publication » (*publishing framework*). Ce dernier prend en charge les transformations dynamiques et la fourniture des résultats au poste client (HTML pour la navigation, PDF pour l'impression).

Les outils annexes. Cocoon se chargera de séquencer et de synchroniser plusieurs traitements. Principalement, il s'agit des transformations du XML en HTML, des équations MATHML en TEX puis leur transformation en images bitmaps. Les transformations s'appuient sur XALAN, un transformeur XSLT, et la gestion des mathématiques sur TEX.

3.1.2. Un outil valant le détour : Cocoon

Il serait trop long de rentrer ici dans tous les détails de cet outils. Vous trouverez plus d'explications à l'adresse `http://xml.apache.org/cocoon`.

Cocoon est un environnement de publication entièrement écrit en Java. C'est pour cela qu'il doit s'appuyer sur un moteur JSP (*Java Server Page*) comme Tomcat. Il exploite les spécifications du W3C[1] comme DOM, XML et XSL pour servir dynamiquement du contenu sur le WEB. Il permet d'atteindre l'objectif tant recherché : la séparation du contenu, de la présentation et de la logique.

Comment ça marche. Le serveur est informé que toute requète de consultation de fichier portant une certaine extension (`.xml` dans notre cas) devra être transmise à Cocoon. Suite à la redirection de la requète, Cocoon lit, sur le serveur, le fichier XML en le parsant (opération consistant à vérifier sa validité).

Une information importante est donnée dès les premières lignes du fichier XML :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<?xml-stylesheet href="these.ex.xsl" type="text/xsl" media="explorer"?>
<?xml-stylesheet href="these.ne.xsl" type="text/xsl" media="netscape"?>
<?xml-stylesheet href="these.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
```

Ces quelques lignes informent Cocoon qu'il va devoir procéder à une opération de type transformation XSLT. Dans le cas où la requète proviendrait d'un navigateur de type Internet Explorer, il faudrait invoquer la procédure XSLT contenue dans le fichier `these.ex.xsl`, dans le cas de Netscape il faudrait se servir de `these.ne.xsl` et dans tous les autres cas de `these.xsl`. Il connaît ainsi les identifiants des navigateurs principaux (dont les Waps).

Quel que soit le type de transformation qui lui est demandé, Cocoon l'exécute et envoie le résultat au poste client. Ce résultat sera le plus souvent un flux HTML ou PDF posté directement sur le port HTTP. C'est en cela que la publication est dite dynamique.

Voilà, c'est tout, du moins pour ce qui nous préoccupe. En fait, Cocoon va beaucoup plus loin. Il gère les transformations XSL:FO avec FOP (un outil prenant du FO et générant du PDF. Il intoduit le concept des XSP (*eXtensible Server Page*) permettant d'aller encore plus loin dans la séparation du contenu et de la logique de traitement.

1. `http://www.w3c.org`

Vous aurez compris que Cocoon n'est, dans l'affaire, qu'un bureau de poste évolué (mais alors très évolué !). La véritable puissance du système réside dans les transformations XSLT

3.1.3. Une phase clé : la transformation

C'est dans la transformation que réside le cœur du système. Elle est responsable de la présentation des résultats mais aussi de l'ergonomie des pages que l'on souhaite obtenir.

Il faut garder à l'esprit qu'en phase de publication, le fichier XML est totalement « neutre ». Le fichier XSLT décide de ce qu'il faut faire du contenu XML. L'écriture d'un tel fichier est une tâche relativement technique mais elle est amortie sur le nombre de documents publiés (puisqu'on ne l'écrit qu'une fois pour une même classe de documents). Cette phase peut être comparée à l'écriture d'une feuille de style LaTeX.

Pour rendre tout cela possible, il est important de maîtriser la structure des documents. Il faut qu'ils se conforment à une structure parfaitement prévisible. La disponibilité d'une DTD, si elle n'est pas obligatoire, peut aider à s'en assurer.

Voici un exemple de source XML automatiquement mis en ligne[2].

```
<p>Aujourd'hui en calcul
de structures la méthode des déplacements est généralement
appliquée aux problèmes statiques de la mécanique.
Les n équations régissant l'équilibre du système se mettent
sous la forme :</p>
<displayequation tag="(1.1)">
<math>
    <semantics>
        <mrow>
            <mfenced open="[" close="]">
            <mrow>
               <msup>
                  <mi>K</mi>
                  <mn>2</mn>
               </msup>
               <mo>(</mo>
                 <mfenced open="{" close="}">
                        <mrow>
                            <mi>d</mi>
                        </mrow>
```

2. Le document dont est extrait cet exemple est issu du mémoire de thèse d'Adeline BOURDON. Son texte intégral est visible à l'adresse suivante : `http://csidoc.insa-lyon.fr/these/1997/bourdon/index.html`. Il nous a été initialement confié pour test sous forme de fichier RTF

```
                </mfenced>
              <mo>)</mo>
            </mrow>
            </mfenced>
            <mo>.</mo>
            <mfenced open="{" close="}">
                <mrow>
                    <mi>d</mi>
                </mrow>
            </mfenced>
            <mo>=</mo>
            <mfenced open="{" close="}">
                <mrow>
                    <mi>F</mi>
                </mrow>
            </mfenced>
        </mrow>
    </semantics>
</math>
</displayequation>
```

L'affichage de ce fragment est présenté sur la figure suivante.

Un fichier XML dynamiquement publié sur serveur WEB.

Sur la partie gauche de la fenêtre, le cadre (*frame*) de table des matières est entièrement pris en charge par la feuille de style de transformation. Elle se charge de son implémentation graphique, mais également de sa constitution logique et de la création des liens. La table des matières ainsi que la spécification des liens hypertextes n'apparaissent à aucun endroit dans le fichier XML de contenu mis en ligne.

Dans cette fenêtre il est possible de rajouter un bouton permettant à l'utilisateur de demander une version imprimable. Cocoon demande alors à XALAN d'effectuer la transformation en LaTeX (sur un style classique ou personnalisé), convertit le résultat en fichier PDF et le sert au poste client.

Il est également possible de rajouter deux boutons permettant de controler la résolution (et donc la taille) des formules pour un confort optimal de lecture[3].

Il serait également envisageable de disposer un bouton permettant de demander à ce que la traduction TeX (ou autre) de ce fichier vous soit adressée par mail. Les seules limites que l'on peut imaginer sont données par notre capacité à écrire des transformeurs.

Comment en est-on arrivé là ? Il serait fastidieux d'exposer la feuille de style XSLT. Mais on peut en décrire brièvement les étapes principales.

1. Parcours global du fichier XML pour extraction de la table des matières. Du code HTML (dans le cadre de gauche) sera spécifiquement généré pour elle. La gestion de la navigation est également faite à ce stade.
2. Parcours global pour créer, pour chaque élément `<math></math>` (les parties MATHML), des formules en TeX à partir desquelles on fabriquera des images GIF.
3. Parcours séquentiel du fichier XML (bien qu'il soit délicat de parler de séquentialité en matière de XSLT) et génération de HTML. À ce stade, il est possible de décider si l'on souhaite ou non utiliser CSS. Dans cette étape, les éléments `<math></math>` provoqueront simplement la génération du code HTML de placement des images des formules. Les valeurs de correction de positionnement par rapport à la ligne de base ont été fournies par le traitement TeX de l'étape 2.

4. Mise en ligne des fichiers issus de RTF4XML

Le XML *flat* généré par RTF4XML correspond en quelque sorte à la DTD du format RTF. Or on a vu que pour se placer dans le cadre d'une publication dynamique, il était nécessaire de disposer d'une instance XML clairement structurée.

3. Ces options n'apparaissent pas sur cette copie d'écran pour des raisons de délai de parution de cette publication.

Partant de l'hypothèse (et donc de l'obligation) que le fichier RTF ait été saisi selon une feuille de style bien identifiée et qu'une certaine cohérence ait été respectée, il est alors possible d'envisager un passage en DTD ou de façon plus modeste (mais souvent suffisante) une mise en structure automatique.

Concrètement, cela signifie que si RTF4XML traite un style RTF `heading 1` par la génération XML suivante :

```
<p StyleName="heading1">Titre de la section</p>
<p StyleName="Normal">Texte...</p>
<p StyleName="Normal">Texte...</p>
```

il faudrait, pour être exploitable, avoir plutôt quelque chose du type :

```
<section>
<title>Titre de la section</title>
<body>
corps_de_toute_la_section
</body>
</section>
```

Corps_de_toute_la_section contenant elle-même des sous-sections, des paragraphes, des tableaux, des figures, des formules centrées, etc.

Chaque élément structurel peut ainsi être décomposé en sous-éléments obligatoires ou facultatifs le constituant. C'est schématiquement le principe d'une DTD.

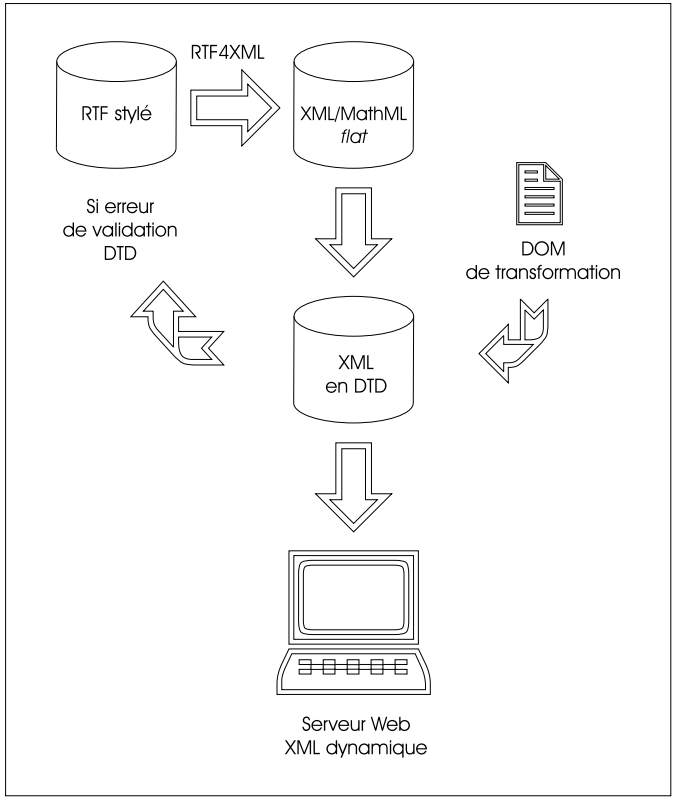
Après avoir dégagé toutes les règles (de bon sens) du type « une section débute sur un style paragraphe de type `heading 1` et se poursuit jusqu'à ce qu'on atteigne un style représentant un titre de chapitre, un titre de section ou la fin du document »[4] il est possible d'écrire un programme réalisant la transformation. Pour cela, XSLT pourrait être envisagé mais serait très délicat à coder. Nous avons donc préféré une approche de transformation consistant à exploiter les API DOM avec Java ou Perl. Il serait envisageable (et certainement préférable) d'utiliser une interface graphique qui exhiberait la structure extraite du fichier RTF, laisserait l'utilisateur spécifier graphiquement et interactivement la transformation, et générerait automatiquement le programme de conversion. En attendant de la développer, on conçoit bien que ce travail est certainement démesuré s'il s'agit de ne gérer que quelques pages d'un document atypique.

4. Exemple volontairement sommaire.

Voilà ! le document XML *flat* est transformé en une instance parfaitement structurée. Cette dernière est alors directement exploitable selon le schéma exposé précedemment.

La mise en place étant faite, les procédures deviennent entièrement automatiques… sauf si la cohérence initiale dans les fichiers RTF n'est pas respectée. Dans ce cas, le système peut continuer à fonctionner mais on risque d'avoir des surprises lors de la publication (WEB et dans une moindre mesure papier). Il faut alors revenir dans le fichier incriminé et le corriger.

Dans le cadre d'une publication massive, il va de soi qu'il n'est pas envisageable de vérifier humainement toutes les publications. L'utilisation d'une DTD devient alors indispensable. Elle permettra, immédiatement après la transformation XML *flat*/XML sur DTD de vérifier que toutes les attentes ont été satisfaites et que la structure est parfaitement respectée.



Synoptique de publication dynamique avec RTF4XML

5. Conclusion

Tant au niveau de la publication traditionnelle qu'au niveau des nouvelles technologies nous sentons bien que XML tend à atteindre les objectifs qu'il s'était fixé.

Il faut reconnaître que la mise en place d'une solution basée sur XML passe par le développement d'une « application XML ». Ceci entend la mise en place de DTD (ou au moins la spécification claire d'une structure) et surtout l'écriture de traitements XSL(T) voire DOM. Ceci fait, l'utilisateur est alors placé dans un contexte de publication robuste, polyvalent et productif. La séparation des compétences nécessaires correspond à la même séparation (contenu, présentation, logique) qu'offre intrinsèquement XML. D'aucun pourrait s'en inquiéter, mais les nouvelles technologies n'ont-elles pas amené de nouveaux métiers (WebDesigner, WebMaster, etc.). Qu'est-il plus intéressant pour un enseignant, un chercheur, un ingénieur : passer du temps sur ce qu'il veut dire ou avec quoi et comment il doit le faire ? XML va leur offrir le moyen de « spécifier des documents » au lieu de simplement les composer.