# Transformation de documents XML avec XSLT

Christophe Riolo

2008-06-17

**Résumé**

Les documents XML prennent une place croissante dans le domaine de l'échange et la sauvegarde de données, aussi bien par internet qu'entre applications, et il apparaît qu'il est nécessaire de pouvoir manipuler ces documents. Le XSLT est un langage XML qui permet de transformer des documents XML en d'autres documents XML, par exemple, simplifier une page XHTML, destiné à un navigateur internet normal, en une page WAP.

Le but de mon TIPE est de programmer en Caml un processeur XSLT, qui prend en entrée le document XML à transformer et la feuille de style XSLT correspondante, et qui applique la transformation.

## Table des matières

# 1 Présentation du XML

## 1.1 Structure globale

Le XML (eXtensible Markup Language, langage balisé extensible) est, comme son nom l'indique, un langage très souple dans sa structure. Il consiste en des balises, qui ont un nom et des arguments, écrits entre chevrons (< et >), qui délimitent, pourrait-on dire, des parties du document. En effet, les balises sont constituées d'une *balise ouvrante* (e.g. `<html>`) et d'une *balise fermante* (e.g. `</html>`). Ce / est caractéristique des balises fermantes. La balise ouvrante et la balise fermante devant se suivre dans cet ordre, il y a donc un *avant la balise*, un intérieur et un après la balise.

Les balises se trouvant à l'intérieur d'une balise données sont ses *fils*, comme pour les arbres (ceci n'est pas dû au hasard, comme nous le verrons). Si une balise n'a pas de fils, elle est dite *vide*, et peut alors être contractée sous la forme suivante : `<nom />`, ce qui est équivalent à `<nom></nom>`.

Ensuite, en XML, les balises doivent être bien imbriquées, c'est à dire que lorsque l'on ferme une balise, tous ses fils doivent être fermés. Voilà un exemple de code bien imbriqué, à comparer avec du code mal imbriqué donné ensuite :

```
<html>
<head />
<body>
Hello world !
</body>
</html>
```

FIG. 1: XML bien imbriqué

Ce code est valide, mais les suivants ne le sont pas car la balise `<head>` est mal fermée :

```
<html>
<head>
<body>
Hello world !
</body>
</html>
```

(a) <head> non fermée

```
<html>
<head>
<body>
</head>
Hello world !
</body>
</html>
```

(b) <head> mal fermée

FIG. 2: XML mal imbriqué

Enfin, le XML n'impose presque pas de contrainte sur les noms des balises, leurs arguments et leurs fils. Il est possible de définir une structure au document avec une *DTD* (Document Type Definition, définition du type de document). Ainsi, il est possible de se fixer des règles précises de syntaxe et, en d'autres termes, de définir un nouveau langage ayant les mêmes caractéristiques que le XML, mais restreintes aux règles fixées. On dira qu'on défini *un langage basé sur XML*.

### 1.2 Applications

De tels langages sont fréquents, même s'il faut être initié au XML pour savoir que ce sont des langages XML et ce que cela signifie.. Il faut aussi savoir que ces langages, en plus d'être nombreux, ont des applications très diverses. Par exemple, tous les documents Open Office sont en réalité des archives de documents XML, c'est donc que l'on peut faire du texte, des feuilles de calcul, du dessin vectoriel, et tout ce que propose Open Office. Rien que ça. Mais d'autres formats existent, ayant différents buts :

- Dessin vectoriel (format SVG : scalable vector graphics)
- Graphismes 3D :
  - de réalité virtuelle (VRML : virtual reality modelling language)
  - interne à Dassault Systèmes, pour les logiciels de modélisation mécanique (3D XML)
- Représentation mathématique (MathML)
- Programmation :
  - XSLT, dont nous reparlerons
  - XUL (XML-based User interface Language), qui permet de développer des extensions firefox et thunderbird
- Communication :
  - protocole XMPP pour la messagerie instantanée Jabber
  - flux de syndication Atom et RSS
  - WML (wireless markup language), qui est le protocole du WAP
  - et bien entendu XHTML

Pour l'anecdote, à propos du XHTML, la syntaxe du XML s'est inspirée de celle du HTML, en la rendant plus rigoureuse, et le XHTML est le pendant XML du HTML. À terme, il le remplacera totalement, et c'est pourquoi le HTML n'évolue plus maintenant, et reste à sa version 4. Bien entendu, cette liste est loin d'être exhaustive, et la flexibilité du XML fait que d'autres langages XML peuvent être développés en interne à un logiciel (comme XUL) ou à une entreprise (comme 3DXML). Cette diversité rend la connaissance du XML capitale pour suivre l'évolution actuelle des technologies de l'information.

## 2 Présentation du XSLT

Le XML est donc un langage très polyvalent, mais pourtant, dès que l'on veut l'afficher, il n'est pas toujours aussi lisible qu'on le souhaiterait, surtout lorsque le fichier n'est pas du SVG, du XHTML ou d'autres langages XML prédéfinis, et il faudra alors définir comment l'afficher.

### 2.1 But du langage

Le sigle XSLT signifie XML Stylesheet Language Transform. Avec le XSL-FO (Xml Stylesheet Language Formatting), il dérive du XSL, qui a la base faisait ce que les deux langages précédemment cités font actuellement : il permettait de mettre en forme un document XML afin de l'afficher. Pourquoi alors l'avoir séparé en deux langages ? Tout simplement parce que si le but est le même, le moyen est différent.

Le XSL-FO permet de décrire des règles d'affichage : "tel élément sera affiché en gras dans un cadre", "tel élément sera affiché en rouge si tel attribut a telle

valeur" ou encore "tel élément ne sera pas affiché du tout".

Le XSLT lui permet de transformer un document XML en un autre document, généralement en un autre document XML, en lui disant "tel élément sera remplacé par tel et tel élément" ou bien "on parcourt tous les éléments, on les classe en ordre alphabétique et on les affiche".

Telle est la transformation, la variation au sein du document, qu'apporte le XSLT. Pour mettre en oeuvre cette transformation, il faut pouvoir repérer précisément les balises, c'est le but du langage XPATH. Sans trop développer le sujet, le langage XPATH permet d'écrire des *chemins* vers des balises, selon la même syntaxe que pour les URL ou les fichiers sous Unix, mais en proposant aussi de faire des calculs et des tests conditionnels, entre autres, et il est donc possible d'imposer des conditions très restrictives sur les balises à sélectionner.

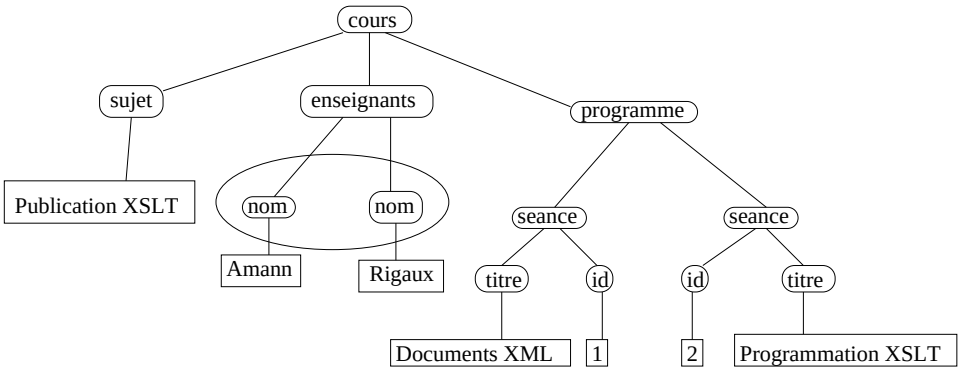


FIG. 3: Balises désignées par l'expression XPATH /cours/enseignants/nom

## 2.2 Exemples de transformations utiles

On peut se demander quel est l'intérêt de faire de telles transformations au lieu d'écrire le document résultant directement. Un intérêt majeur est la stabilité des données écrites. En effet, imaginons un scientifique à qui l'on dit d'écrire ses rapports selon une mise en forme bien précise. Il aura tout intérêt à rédiger tous ses articles en XML, selon toujours la même syntaxe, et de les transformer avec *une unique feuille de style XSLT*. En effet, si jamais la mise en forme doit changer, plutôt que de réécrire tous ses articles, il lui suffirait de modifier la feuille XSLT et de réappliquer les transformations. Cela ne fait qu'une modification à faire, et assure la pérennité des données enregistrées par le scientifique.

Encore mieux, si la transformation est faite à la demande, il n'aura même pas à refaire les transformations, puisqu'elles seront effectuées lorsque quelqu'un voudra lire un article. Bien entendu, il est tout a fait possible que le document

XML soit normalisé (on définit un langage XML, pourquoi pas XMLReport) et alors tout le monde pourrait utiliser la même feuille de style XSLT à travers le monde.

Mais l'intérêt ne s'arrête pas là. Supposons que notre scientifique fasse des expériences et s'intéresse à des statistiques. On supposera qu'il dispose les résultats dans un fichier XMLStats (langage imaginaire), ou pourquoi pas, à partir du tableur Open Office. Il voudra pouvoir mettre ses résultats sous différents formats, par exemple, un document web, des graphiques, un fichier .tex et un PDF.

Alors pour *toutes* ses expériences, il ne lui faudra écrire qu'un seul fichier XMLStats, qu'il transformerait avec différentes feuilles de style XSLT : XMLStats -> XHTML, XMLStats -> SVG, XMLStats -> .tex (on peut transformer du XML en texte avec XSLT) et XMLStats -> XSL-FO -> PDF.

L'intérêt est alors très visible : tout est automatisé à partir d'un simple fichier, éventuellement rempli automatiquement par un logiciel de capture. Et à ceci se combine encore le premier avantage, à savoir la pérennité des données, et l'unicité de la feuille de style. Pour dix expériences, cela ne fait que 10 fichiers à remplir au lieu de 40.

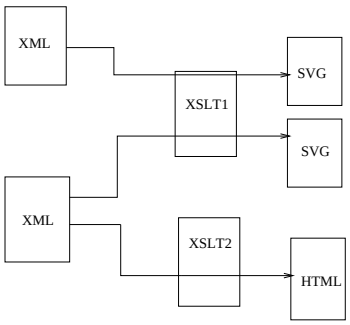


FIG. 4: En XSLT : un pour tous et tous pour un !

# 3 Présentation de l'algorithme de la transformation

Nous avions vu à la section 1.2 que le XSLT est un réalité un langage de programmation. Comme un fichier XSLT est un fichier texte, c'est donc nécessairement un langage interprété, et il faut donc un interpréteur, appelé *processeur XSLT*, prenant en entrée le document XSLT, le document XML a transformer et renvoyant le résultat de la transformation. J'ai étudié les mécanismes de la transformation XSLT afin de trouver comment programmer en Caml un processeur XSLT.

## 3.1 Équivalence XML / arbre : analyse et sérialisation

Tout d'abord, il est impératif de remarquer qu'un document XML est rigoureusement équivalent à un arbre dont les noeuds sont les balises, les arguments et le texte normal (ainsi éventuellement que quelques noeuds spéciaux comme

les commentaires, que je n'ai pas implémentés). Le XML a été créé pour et cette équivalence est mentionnée dans ses spécifications. Le travail sur un document XML est donc rigoureusement le même que le travail sur un arbre. C'est pourquoi le processeur XSLT commence par transformer les documents XML et XSLT en arbres, procède à la transformation de l'arbre XSLT (car c'est le document XSLT qui contient la structure du document sortie) pour obtenir arbre correspondant au document résultant, qu'il transforme ensuite en texte.

La transformation texte -> arbre est appelée *analyse*, et la transformation arbre -> texte *sérialisation*

Deux choix technologiques sont alors possibles pour l'analyse : les expressions rationnelles ou la décomposition en lexèmes par l'analyseur lexical, et sont équivalentes. J'ai personnellement opté pour l'analyseur lexical pour des raisons d'habitude. Le principe est le même : transformer le document texte en un flux d'éléments (de lexèmes pour l'analyseur lexical, de balises ouvrantes ou fermantes, d'arguments et de texte pour les expressions rationnelles), et d'utiliser les possibilités de Caml sur les filtrages de flux pour construire l'arbre, en regardant ce que vaut chaque élément tour à tour.

La sérialisation est beaucoup plus simple, et consiste à parcourir l'arbre et à écrire les éléments rencontrés au bon endroit.

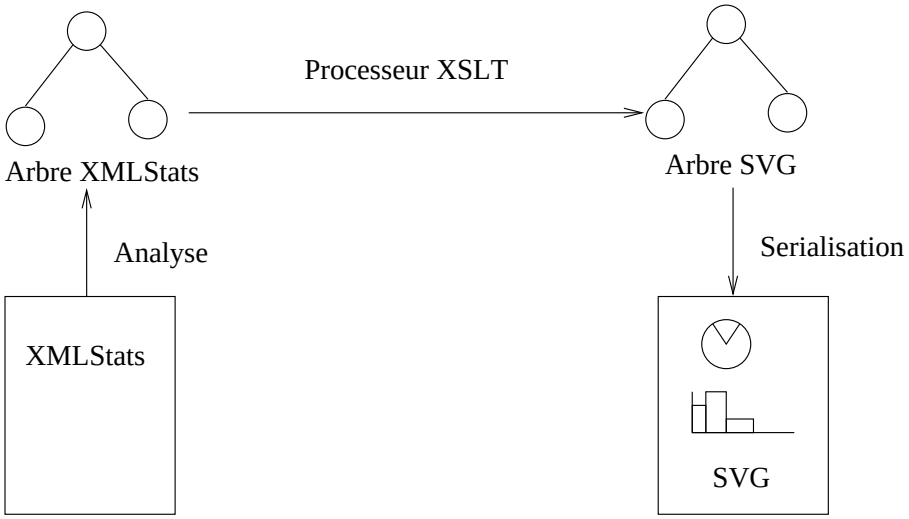


FIG. 5: Exemple de transformation XSLT

## 3.2 Parcours de l'arbre XSLT et transformation

La véritable difficulté dans la programmation du processeur et qu'en réalité, il doit parcourir deux arbres simultanément. En effet, les balises XSLT ont besoin d'un *noeud contexte*, c'est à dire en réalité d'un noeud à partir duquel sont évaluées les expressions XPATH.

Par exemple, <value-of select="."> sélectionne la valeur texte du noeud contexte, nous voyons en image ci après l'importance du noeud contexte.

Pour cela, j'ai opté pour un représentation de l'arbre où les fils sont dans un **tableau** et non d'une liste. L'intérêt est qu'un noeud est représenté très précisément par la liste des indices de tableau de ses ancêtres, et de lui même. Ainsi il suffit de garder en référence l'arbre XML et l'arbre XSLT, et de localiser les noeuds contexte (XML et XSLT) par la liste des indices en question.

Cette difficulté surmontée, il s'agit de parcourir l'arbre XSLT, de laisser les noeuds qui ne sont pas des balises XSLT telles quelles, et sinon d'appliquer la transformation idoine (ce qui n'est toujours pas simple à programmer), le noeud contexte XML étant connu. Pour cela, j'ai du programmer un fonction qui extrait de l'arbre XML toutes les balises vérifiant une expression XPATH, sous la forme d'un tableau, ce qui simplifiait grandement le processus.

# 4 Conclusion

Pour ce TIPE, j'ai pu approfondir ma connaissance du XML et des transformations XSLT, et même si le processeur que j'ai programmé est limité, car n'intégrant ni toutes les possibilités de XPATH ni celles de XSLT, mais les mécanismes utilisés sont plus génériques que leur mise en oeuvre technique. Mais le point important n'est pas la programmation d'un processeur XSLT, car il en existe d'excellents, comme Xalan, de la fondation Apache, mais bien de mieux comprendre les enjeux et le fonctionnement de telles transformations.

# Références


[1] Amann Bernd et Rigaux Philippe,*Comprendre XSLT*, édition O'Reilly, 2006

[2] Leroy Xavier et Weis Pierre, *Manuel de référence du langage Caml*, Inter-Editions, 1993

[3] site d'apprentissage et de référence des technologies web, *http ://www.w3schools.com/*