

# Mise en œuvre de ce blog

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 2 Novembre 2005. Dernière mise à jour le 16  
Février 2006

<http://www.bortzmeyer.org/blog-implementation.html>

---

Ce blog utilise quelques techniques originales, autour de XML, et cet article les décrit. Il vise un public d'informaticiens. Si seul le contenu vous intéresse, vous serez sans doute déçus mais vous pouvez aller voir, dans ce cas, mon article sur « Pourquoi et comment je blogue <<http://www.bortzmeyer.org/metablogging.html>> ».

Revenons d'abord sur le cahier des charges technique, sans lequel on ne peut pas comprendre les choix effectués. Je voulais un blog simple, sans fonctions sophistiquées, je ne cherchais pas à avoir d'interactivité (le lecteur ne peut pas laisser de commentaires, ce qui me convient <<http://www.bortzmeyer.org/no-comment.html>>), par contre, il était impératif pour moi de pouvoir publier sur n'importe quel serveur Web, sans exigences sur la disponibilité de tel ou tel programme, et il était également nécessaire de pouvoir travailler sur n'importe laquelle de mes machines, même sans accès au réseau. D'autre part, je ne voulais utiliser que du logiciel libre, ce qui élimine la plupart des programmes écrits en Java : il existe des mises en œuvre libres du langage Java mais peu de programmes Java réels arrivent à tourner dessus.

Sur le moment, je n'avais pas trouvé de système correspondant à ce cahier des charges. Par la suite, Sophie-Charlotte Barrière m'a indiqué PyBloxom <<http://pybloxom.sourceforge.net/>> qui semble en effet convenir. Mais trop tard, j'ai développé ma solution. (Depuis, il y a encore eu d'autres solutions comme Jekyll <<http://jekyllrb.com/>> ou Hyde <<http://ringce.com/hyde>>.)

Le pivot des techniques utilisées est XML. L'un des intérêts d'un langage formel comme XML est que cela permet de calculer facilement des statistiques <</auto/summary.html>>. Tous les articles ont été rédigés en XML, dans un langage spécifique à ce blog. XML est en effet un méta-langage, où la syntaxe est spécifiée, mais pas le vocabulaire (la liste des **éléments**) ou la grammaire (les relations possibles entre éléments). Pour décrire le vocabulaire et la grammaire acceptable, on utilise un langage dit "de schéma" comme DTD ou bien Relax NG, que j'utilise.

Voici donc une partie de la spécification du langage utilisé pour ce blog, dans la syntaxe compacte de Relax NG (le programme Trang <<http://www.thaiopensource.com/relaxng/trang.html>> peut convertir entre la syntaxe compacte et la syntaxe XML) :

```
# Date of first writing
date = element date {xsd:date}
...
content = element content {paragraph+ & code*}
...
entry = element entry {entry.attlist & date & update_date? & summary? & content}
title.attrib = attribute title { text }
entry.attlist &= title.attrib
```

Une fois le schéma spécifié, on peut écrire des fichiers XML et tester leur validité.

```
<entry title="Mise en &#x153;uvre de ce blog">
<date>2005-10-30</date>
<content>

<p>Ce blog utilise quelques techniques originales, autour de XML, et
```

Rappelons qu'il existe deux niveaux de vérification dans XML : un document XML doit être **bien formé**, ce qui signifie qu'il doit obéir à la syntaxe XML (par exemple, toute étiquette de début d'élément doit être fermée par une anti-étiquette correspondante) et il peut être **valide**, ce qui est plus fort : la validité est la conformité à un schéma, ici écrit en Relax NG.

J'utilise l'outil **xmllint** (qui fait partie de libxml2, la bibliothèque XML du projet GNOME, mais qui peut s'utiliser en dehors de GNOME), pour tester la validité des fichiers avant toute publication :

```
xmllint --noout --relaxng ../schemas/entry.rng blog-implementation.entry_xml
```

. Mais je vais peut-être changer pour **rnv** <<http://www.davidashen.net/rnv.html>> (jing, lui, ne tourne que sur du logiciel non-libre).

Le traitement des fichiers XML est ensuite réalisé par des programmes XSL. On les appelle souvent des "feuilles de style" mais c'est un terme inadapté, XSL étant un vrai langage de programmation. Conçu pour traiter du XML, XSL réalise la traduction en HTML pour publication sur le Web ou bien en texte seul pour envoi par le courrier. Un exemple de programme (partiel) XSL qui transforme un élément "rfc" en texte "RFC nnn" tout en mettant un lien vers le serveur officiel en [www.ietf.org](http://www.ietf.org) :

```
<xsl:template match="rfc">
  <xsl:variable name="num">
    <xsl:value-of select="@num"/>
  </xsl:variable>
  <a href="http://www.ietf.org/rfc/rfc{ $num }.txt">
    <xsl:text>RFC </xsl:text><xsl:value-of select="@num"/>
  </a>
</xsl:template>
```

J'utilise le processeur XSL **xsltproc** (qui fait partie de libxml2, la bibliothèque XML du projet GNOME, mais qui peut s'utiliser en dehors de GNOME), pour effectuer la transformation (j'ai aussi testé Sablotron <[http://www.gingerall.com/charlie/ga/xml/p\\_sab.xml](http://www.gingerall.com/charlie/ga/xml/p_sab.xml)>, qui fonctionne bien) :

---

<http://www.bortzmeyer.org/blog-implementation.html>



```
def date_of(dom):
    match = xpath.Evaluate("//date/text()", dom)
    if match:
        return match[0].nodeValue
    else:
        raise Exception("No date in the DOM tree")
```

Toutes ces opérations sur les fichiers sont coordonnées par `make`, un programme qui prend en entrée une description des tâches à faire, le `Makefile` et qui exécute ces tâches dans l'ordre nécessaire. Le langage d'écriture des `Makefile` est un langage **déclaratif** où on indique les pré-conditions et les actions d'une tâche et où `make` trouve l'ordre d'exécution.

`make` permet donc très simplement de créer les pages HTML, le fil ATOM et de les publier, simplement en tapant `make install`. Voici une partie du `Makefile` de ce blog, où on explique à `make` comment créer une entrée ATOM pour un article :

```
%.atom: %.entry_xml
xsltproc -o $@ \
--stringparam filename `basename $< | sed 's/\.entry_xml$$//'\` \
${ENTRY_ATOM_STYLESHEET} $^
```

Pour éditer les fichiers XML, j'utilise `emacs`, comme pour tous mes autres fichiers texte. `Emacs` permet de charger des **modes** différents selon le langage édité. Il existe plusieurs modes pour XML et j'utilise `nXML` <<http://www.thaiopensource.com/nxml-mode/>>, qui rend très facile l'édition de documents XML valides.

Pour gérer les fichiers et leurs versions successives, je me sers de `darcs` <<http://www.darcs.net/>>, un système de gestion de versions décentralisé, qui rend très agréable le travail sur plusieurs ordinateurs. Je peux travailler au bureau, à la maison, ou bien avec mon portable sans avoir à penser à copier les fichiers. `darcs` se souvient de ce qui a été fait et, sur chaque machine, je peux voir facilement où j'en suis. Voici par exemple le résultat de la commande `darcs changes` qui affiche l'historique du dépôt `darcs`.

```
Sun Oct 23 13:49:16 CEST 2005  stephane@ludwigVI.sources.org
* CPL termine + bogue dans la feuille XSL

  ./entries/cpl-maison-NOT-YET.entry_xml -> ./entries/cpl-maison.entry_xml
M ./entries/cpl-maison.entry_xml -2 +2
M ./schemas/common.xsl -1 +1

Sat Oct 22 09:04:53 CEST 2005  stephane@ludwigVI.sources.org
* RFC 4192 publié

  ./RFC/4192-NOT-YET.rfc_xml -> ./RFC/4192.rfc_xml
M ./RFC/4192.rfc_xml -1 +1
```

Si vous voulez utiliser les mêmes techniques, ou simplement les étudier plus en détail, n'hésitez pas à récupérer les fichiers complets (en ligne sur <http://www.bortzmeyer.org/files/blog-support-files.tar.gz>), ils sont distribués sous licence libre (GPL). **Attention** : ils sont peu documentés, c'est un système développé uniquement pour ce blog et qui n'est donc pas utilisable tel quel chez vous.