Détails techniques sur l'écriture de mon livre « Cyberstructure »

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 12 décembre 2018

https://www.bortzmeyer.org/livre-tech.html

Comme vous avez pu le voir dans un autre article <https://www.bortzmeyer.org/livre-publie.html>, j'ai écrit un livre nommé « Cyberstructure » et qui parle des relations entre l'architecture technique de l'Internet et les questions politiques. Ce nouvel article est destiné uniquement aux détails techniques de l'écriture, pour ceux et celles qui se demandent « tu as utilisé quel logiciel pour faire ce livre ? ». Je ne parlerai donc pas ici du contenu du livre.

D'abord, pour les outils utilisés, il faut bien voir que l'auteur n'a pas une liberté complète puisqu'un livre est un travail collectif. Il faut donc une discussion avec l'éditeur, du moins si, comme moi, on a un éditeur qui se penche sur le texte, au lieu de demander à l'auteur des images qu'on imprimera telles quelles. D'autre part, dans mon cas, la mise en page était faite par l'éditeur (avec InDesign), donc je n'avais pas à me soucier du rendu, je pouvais me concentrer sur le texte. Enfin, certains choix ne concernaient que moi, puisqu'ils ne changeaient rien à ce qui était échangé avec l'éditeur.

Les choix importants, après cette discussion, étaient :

— N'utiliser que du logiciel libre,
— Format XML, avec un schéma développé par moi,
— Utilisation de l'éditeur Emacs et de son mode XML nxml-mode <https://www.emacswiki.org/emacs/NxmlMode>,
— Utilisation du VCS darcs.

Pourquoi ces choix ? Commençons par le format XML. C'est un format simple pour l'auteur, bien adapté au texte (contrairement à JSON) grâce notamment à la possibilité de mélanger élements structurés et texte, comme par exemple :

```
<p>L'ARJEL, l'autorité de régulation des jeux en ligne, a
été, sauf erreur, la première autorité ayant ce droit de censurer,
sur la base du décret <cite url="https://www.legifrance.gouv.fr/eli/decret/2011/12/30/BCRB1120950D/jo/texte">n°
modalités d'arrêt de l'accès à une activité d'offre de paris ou de
jeux d'argent et de hasard en ligne non autorisée</cite>. [...]
C'est ainsi que des sites distribuant des fichiers
« torrent » <ref target="bittorrent">pour une explication</ref>
comme The Pirate Bay ou T411 ont fait l'objet de décisions de justice
imposant leur blocage.</p>
```

Et, contrairement à JSON, on peut y mettre des commentaires, ce qui aide beaucoup l'auteur au long des mois de réécriture et de modifications. La discussion avec l'éditeur a permis de s'assurer qu'InDesign pouvait importer du XML sans mal, et le choix de XML a donc été en partie guidé par l'éditeur. (Par exemple, il n'y avait pas de moyen simple d'importer du LaTeX. LaTeX repose sur un langage de programmation complet, qui est donc difficile à importer, sauf à utiliser le moteur TeX. XML, au contraire, ce ne sont que des données, sans programme.)

La mise en page étant faite par l'éditeur, il était important que je me limite au marquage sémantique. Par exemple `<p>Le RFC 7962, <work xml:lang="en" url="https://www.rfc-editor.org/info/rfc7 Network Deployments: Taxonomy, Characterization, Technologies, and Architectures</work` sans préjuger de comment serait rendu le titre du document cité : on indique que c'est un titre, on indique la langue, et la personne qui fera la mise en page pourra suivre les bonnes pratiques de la mise en page, indépendamment du contenu. Les éléments XML possibles et leurs relations sont mises dans un schéma, écrit en Relax NG. Ce schéma est conçu uniquement pour ce livre, et je n'ai pas cherché à le faire beau ou propre ou général. Si vous voulez le voir, il est dans le fichier (en ligne sur `https://www.bortzmeyer.org/files/livre.rnc`). Je teste la conformité du texte au schéma avec rnv `<http://www.davidashen.net/rnv.html>` :

```
% rnv livre.rnc livre-noent.xml
%
```

Un autre avantage de XML par rapport à LaTeX, dans ce contexte, est qu'il est facile de développer des outils traitant le XML et effectuant certaines opérations. Par exemple, j'ai fait un programme XSLT pour n'extraire que le texte du livre, afin de compter caractères et mots. En revanche, LaTeX est certainement imbattable quand il faut faire une jolie sortie PDF ou papier sans trop y passer de temps. Et la plupart des relecteurs, à commencer par moi, ont préféré travailler sur cette sortie que sur le source XML. Pas de difficulté, encore un autre programme XSLT, pour convertir le XML en LaTeX, qui était ensuite traité. Voici ce programme : (en ligne sur `https://www.bortzmeyer.org/files/tolatex.xsl`), mais rappelez-vous que ce n'est pas lui qui a été utilisé pour le rendu final du livre. (Pour faire tourner ce programme XSLT, j'ai utilisé xsltproc `<http://xmlsoft.org/XSLT/xsltproc.html>`, dans la libxslt. Au passage, certains des outils étaient d'usage compliqué donc j'ai utilisé le classique make pour orchestrer leur exécution.)

Notez que les techniques utilisée pour le livre étaient assez proches, voire identiques, à celles de mon blog ce qui n'est évidemment pas un hasard. Les techniques du blog sont déjà documentées `<https://www.bortzmeyer.org/blog-implementation.html>`.

Apparemment, la plupart des auteurs de livres utilisent plutôt un gros cliquodrome `<https://fr.wiktionary.org/wiki/cliquodrome>` comme Word ou LibreOffice. Mais je n'aime pas ces logiciels (j'avais déjà critiqué leur approche afterword `<https://www.bortzmeyer.org/afterword.html>` il y a plus de dix-sept ans.)

Le XML a l'avantage, comme JSON ou LaTeX, d'être un format texte, donc qui peut être traité avec tous les outils existants. C'est par exemple le cas du choix de l'éditeur (l'éditeur de textes, pas l'éditeur du livre). Pas besoin de concertation, cette fois, ce choix de l'éditeur de textes est purement local et n'affecte pas ce qui est envoyé à l'éditeur du livre. J'ai donc utilisé mon éditeur préféré, emacs. On peut écrire du XML avec le mode de base d'Emacs mais ce n'est pas très amusant (taper le début de l'élement XML, sa fin, penser à bien fermer tout ce qui a été ouvert), il vaut donc mieux utiliser un mode Emacs adapté au XML. J'ai utilisé nxml-mode `<https://www.emacswiki.org/emacs/NxmlMode>`. À part le fait qu'il économise du temps de frappe, et qu'il affiche le source XML proprement coloré (les noms des éléments en bleu, les commentaires en rouge, pour les distinguer du texte), le gros avantage

de nxml-mode est qu'il connait Relax NG, et qu'une fois configuré pour utiliser mon schéma, il peut guider l'écriture, indiquant quels sont les éléments XML acceptables à l'endroit où se trouve le curseur, et validant le résultat au fur et à mesure. Grâce à cela, la validation XML complète, faite avec rnv, était quasiment inutile.

Le schéma XML que j'avais fait me semblait raisonnable, avec un usage intelligent des éléments et des attributs XML (un sujet toujours passionnel <https://stackoverflow.com/questions/33746/xml-attribute-vs-xml-element> dans le monde XML). Mais lors de l'importation dans InDesign, un problème est apparu : sauf exception, InDesign ne permet pas de mettre du contenu dans les attributs <https://helpx.adobe.com/fr/indesign/using/structuring-documents-xml.html>. Il a donc fallu transformer plusieurs attributs en éléments, avec le programme XSL (en ligne sur https://www.bortzmeyer.org/files/toindesign.xsl).

Contrairement à mon blog, où certains articles ont été écrits d'une seule traite, ce livre a fait l'objet de retours, de révisions, de critiques et de remords. Il a donc fallu le modifier plusieurs fois, et parfois remettre en place des paragraphes que j'avais effacé quelques jours plus tôt. L'outil idéal pour cela est évidemment le VCS. Comme VCS, j'ai choisi darcs. Il est moins connu que git mais bien plus facile à utiliser. Il ne fournit pas de mécanisme de coopération pratique, mais ce n'est pas grave ici, puisque j'étais seul à travailler sur le texte. Le reproche que j'ai le plus entendu sur darcs est qu'il n'a pas le concept de branches. Cela me parait plutôt un avantage : d'abord, les branches sont très compliquées à utiliser, et ensuite un VCS décentralisé, comme git ou darcs, n'a pas vraiment besoin de branches : il suffit de faire des dépôts différents.

Comme tous les VCS, darcs permet de voir l'historique d'un travail :

```
patch f1c1609a11c27b0a125057f3afe0e91d537f1fdb
Author: stephane@sources.org
Date:   Sun Dec 10 17:54:48 CET 2017
  * Traduction en XML de plan, avant-propos et middleboxes, rédaction de utilisateurs

patch 28b1ff5fef2d627f662d41cabda249bb585fec69
Author: stephane@sources.org
Date:   Sun Dec 10 12:20:17 CET 2017
  * Début de la version XML

patch 7e38de35163eccb6365502ec0ad189dcd3af5446
Author: stephane@sources.org
Date:   Wed Nov 29 10:16:05 CET 2017
  * Questions à l'éditeur

patch 67e7a11537ccc3d5d0b02d4bf83a3ae051bca25d
Author: stephane@sources.org
Date:   Wed Nov 22 16:20:17 CET 2017
  * Article middleboxes

patch 3e71659b7b880f5cad02a441368a0e753b36de15
Author: stephane@sources.org
Date:   Sun Nov  5 17:13:40 CET 2017
  * Début du travail sur le livre
```

Cela permet aussi de voir combien de "*commits*" sont faits :

```
%  darcs    changes . --count
Changes to Livre:
485
```

(Oui, on aurait pu faire `darcs changes . | grep Date | wc -l` .) Comme toutes les métriques quantitatives d'un travail humain, ce chiffre n'a guère de signification ; il dépend de si on "*commite*" souvent ou seulement à la fin de la journée, par exemple. C'est juste amusant.

Un avantage important d'un VCS réparti (comme darcs ou git) est que chaque copie locale est un historique complet du travail. Tout est donc automatiquement réparti sur chaque machine (le PC fixe à la maison, le portable en déplacement, plus une ou deux machines hébergées à l'extérieur, et une clé USB…), ce qui est une forme efficace de sauvegarde. J'ai vu plus d'une fois un étudiant perdre toute sa thèse parce que les fichiers se trouvaient sur une seule machine, en panne, perdue ou volée, pour ne pas avoir envie de faire comme eux. Et l'utilisation du VCS réparti pour cela est moins pénible que la plupart des systèmes de sauvegarde.

J'ai dit plus haut que les métriques quantitatives n'avaient guère de sens ; l'avancement du travail ne se mesure pas au nombre de caractères tapés ! Si ces métriques sont assez ridicules quand des chefs de projet prétendent les utiliser pour suivre le travail de leurs subordonnés, elles sont quand même utiles à l'auteur qui :

— Sait quelle confiance limitée il faut leur accorder,
— A envie de suivre un peu ce qu'il fait.

J'ai donc écrit quelques scripts très simples pour compter quelques trucs que je trouve utiles. Pour compter le nombre de caractères, je me sers d'un programme XSLT, (en ligne sur `https://www.bortzmeyer.org/files/totext.xsl`), qui garde uniquement le texte avant de passer à `wc` (utiliser wc sur le fichier XML compterait en trop toutes les balises XML) :

```
% make count
xsltproc totext.xsl livre-noent.xml > livre.txt
wc -c livre.txt
577930 livre.txt
```

Je marque les choses à faire dans le texte avec la chaîne de caractères « TODO » (à faire). Il y a donc aussi des scripts pour compter ces TODO. Par exemple, `make count` comptait aussi les TODO :

```
% make count
xsltproc totext.xsl livre-noent.xml > livre.txt
wc -c livre.txt
432717 livre.txt
Encore 59 TODO
```

Le source du livre était découpé en plusieurs fichiers, donc il était utile pour moi de savoir quels fichiers avaient le plus de TODO, avec `grep` et `sort` :

```
% make todo
grep -c TODO *xml | grep -v ':0$' | sort -r -n -t: -k 2
neutralite.xml:8
censure.xml:7
technique.xml:6
gouvernance.xml:5
blockchain.xml:5
droitshumains.xml:4
securite.xml:3
plateformes.xml:3
adresse-ip-exposee.xml:3
acces.xml:3
...
```

Il était également utile pour moi de compter la proportion de TODO par fichier (certains sont plus gros que d'autres) :

```
% ./todo-pct.sh
protocoles.xml:3
gouvernance.xml:2
finances.xml:0
...
```

(Oui, le script est disponible ici (en ligne sur `https://www.bortzmeyer.org/files/todo-pct.sh`).)

Le livre bénéficie d'un site Web d'accompagnement, . C'est un site Web statique <https://www.bortzmeyer.org/generateurs-web-statiques.html>, utilisant le générateur de sites statique Pelican <https://getpelican.com/>. Comme promis dans le livre, le serveur HTTP (Apache) n'enregistre pas votre adresse IP ou le type de navigateur Web utilisé. La configuration d'Apache correspondante est :

```
LogFormat "%u %t \"%r\" %>s %O \"%{Referer}i\" %v" minimum

CustomLog /var/log/apache2/access_cyberstructure.log minimum
```

Ce qui donne dans le journal des lignes comme :

```
- [19/Dec/2018:01:33:31 +0100] "GET / HTTP/1.1" 200 6420 "https://www.bortzmeyer.org/livre-publie.html" cyberstr
```

Je n'ai pas utilisé de correcteur orthographique. Il en existe en logiciel libre comme aspell mais je les trouve peu utiles : je fais relativement peu de fautes d'orthographe <https://cyberstructure.fr/errata.html>, j'ai d'excellents correcteurs humains <https://www.bortzmeyer.org/remerciements.html>, et il y a beaucoup de termes techniques dans le livre que le correcteur informatique ne connait pas, ce qui aurait rendu son utilisation pénible. (Il faut ajouter plein de mots à la première utilisation.)

```
% aspell check -l FR livre.txt
```

(Il faut avoir installé le paquetage `aspell-fr` pour avoir les mots français.)

Je ne suis pas bon pour les dessins (disons même que je suis franchement nul). Les schémas ont donc été faits de manière sommaire avec Asymptote, puis refaits proprement par un graphiste professionnel.