XML to paper publishing with manual intervention

Oleg Parashchenko

bitplant.de GmbH
Fabrikstr. 15, 89520 Heidenheim, Germany
olpa@uucode.com

Abstract. Existing tools consider XML to paper conversion as one black box step and provide control only through predefined options. With this approach, tuning the layout of output documents is a burdensome task.
The paper advocates a new workflow for XML to paper conversion, in which a separate step allows the user to fine control the layout. The changes made by the user are remembered and later can be automatically re-applied during publishing an updated version of the document.
A possible technical implementation for the workflow is suggested. TEX is used as a typesetting engine. XML to TEX conversion is made using XSLT and TEXML. The management of changes is performed by diff and patch tools.

1 Introduction

Publishing XML on paper, especially creating books, requires that the resulting layout is perfect. Unfortunately, sometimes the layout is unsatisfactory, regardless how good the conversion scripts are. In such cases human assistance is required.

Manual intervention is possible on three levels: as hints for conversion scripts in the source XML, as corrections in the final format, or somewhere inbetween. The first, putting hints in the source, is the simplest, but it pollutes the logical markup with presentation details. Also, a set of possible hints can be not enough for fine tuning. The second option, correcting layout in the final format (usually PostScript or PDF) is good for local changes, but does not suitable for such tasks as inserting a page break. I think that the last option, tuning somewhere inbetween, could be best.

XSL [1] is the main candidate. To publish XML on paper, W3C recommends the XSL standard: an XSLT program converts a source XML to XSL formatting objects (XSL-FO), and some tool performs actual typesetting as specified. Unfortunately, XSL-FO files are not intended and not suitable for editing by hand, and there are no specialized editors.

An important possibility is importing XML to a desktop publishing program and tuning the layout there. The problem, which also exists in the case of an

imaginary XSL-FO editor, is that if the source XML is changed, then the user should re-import XML again and repeat all the changes.

As a solution, we propose to use the typesetting system TEX [2] as an intermediate between XML and paper. A TEX document is an usual text file, and no special editor is required for it. With some effort, XML can be converted to a good TEX file, such that it does not look like an autogenerated mess. The user can edit this file to tune the layout.

The diff/patch [3] mechanism solves the second problem, the need to repeat the changes in the updated document. In software development, diff and patch tools are used to join changes from different developers. But the tools are not restricted to programming code, they work with any text files. For our needs, the changes in the layout and the changes in the document also can be merged. Effectively, it looks like the user-made changes are automatically applied to the new version of the document.

The idea of using TEX for XML publishing isn't new (see [4]), but all the existing proposals consider XML to paper conversion as one logical step. The user has to use only pre-defined options and is not supposed to interfere with the generation process. The same is for non-TEX software: I'm not aware of tools which encourage the user to create and manage changes.

The rest of the paper describes the complete workflow. Different technical implementations are possible. I propose one based on the standard tools (XSLT, diff, patch), on TEXML [5], a tool for supporting XML to TEX conversion, and finally on the own experience.

2 Workflow

A graphical illustration of the workflow is shown on the Fig. 1. Version numbers are for illustrative purpose, not the real ones. "PDF" means any suitable output format for printing.

1. Convert an XML document version 1.0 to a TEX document version 1.0.0 and then to a PDF document version 1.0.0.
2. Change the layout of the document version 1.0.0. This gives the document version 1.0.1.
3. Repeat the step 2 as long as necessary. The final document version is 1.0.N.
4. Remember the differences between the TEX documents 1.0.0 and 1.0.N in a patch file 1.0.0-1.0.N.
5. If the source XML document is updated to the version 1.1:
   - Generate a TEX document version 1.1.0
   - Apply patch 1.0.0-1.0.N to the generated file, this gives the document version 1.1.N. Restart the workflow at the step 3, using the new version numbers.

The workflow description is high-level. It explains what to do, but doesn't specify how to do it. Different technical implementations are possible, the one used by the author is described in the next section.

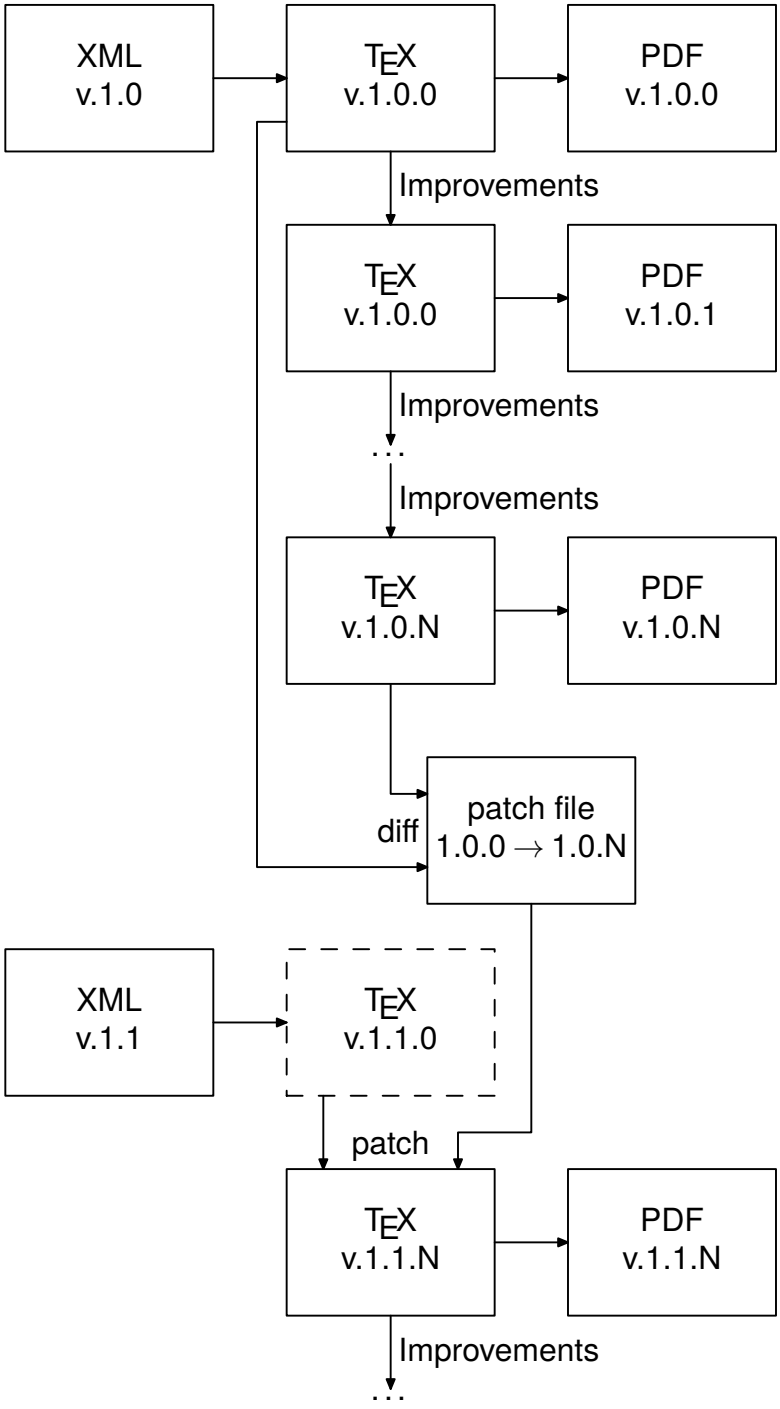


Fig. 1. XML to paper workflow

3 Possible Technical Implementation

3.1 XML to TEX

Different approaches for converting XML to TEX are possible. My personal choice is TEXML: an XSLT program converts XML to an intermediate XML-based format TEXML, and a serializer makes a TEX file from TEXML. This approach is advocated in the article [5]. In short:

- XSLT is the standard for transforming XML,
- Producing the correct TEX syntax is a hard task, better to be delegated to a specialized tool.
- The TEXML serializer automatically makes a human-friendly TEX file with good formatting.

As an example, consider the following source XML file `article.xml`.

```
<article>
<title>De finibus bonorum et malorum</title>
<para>Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
sed diam nonumy eirmod tempor invidunt ut labore et dolore
magna aliquyam erat, sed diam voluptua. ...</para>
<para>Ut wisi enim ad minim veniam, quis nostrud exerci tation
ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo
consequat. ...</para>
</article>
```

An XSLT program `democonv.xsl` to convert the XML to TEXML:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="article">
  <TeXML>
    <xsl:call-template name="generate-header"/>
    <env name="document">
      <xsl:apply-templates/>
    </env>
  </TeXML>
</xsl:template>

<xsl:template match="title">
  <cmd name="title">
    <parm><xsl:apply-templates/></parm>
  </cmd>
</xsl:template>
```

```
<xsl:template match="para">
  <env name="para">
    <xsl:apply-templates/>
  </env>
</xsl:template>

<xsl:template name="generate-header">
  <TeXML escape="0">
\documentclass{article}
\usepackage{democonv}
% ... more document-specific settings ...
  </TeXML>
</xsl:template>

</xsl:stylesheet>
```

The generated TEX file `article.tex` is:

```
\documentclass{article}
\usepackage{democonv}
% ... more document-specific settings ...
\begin{document}
\title{De finibus bonorum et malorum}
\begin{para}
Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
sed diam nonumy eirmod tempor invidunt ut labore et dolore
magna aliquyam erat, sed diam voluptua. ...
\end{para}
\begin{para}
Ut wisi enim ad minim veniam, quis nostrud exerci tation
ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo
consequat. ...
\end{para}
\end{document}
```

This conversion example shows that one can get one-to-one mapping between XML and TEX structural elements. (More precisely, TEX is used here in the form of its macro package LATEX.) Correspondingly, the conversion XSLT-program is trivial.

Formatting is described in the file `democonv.sty`:

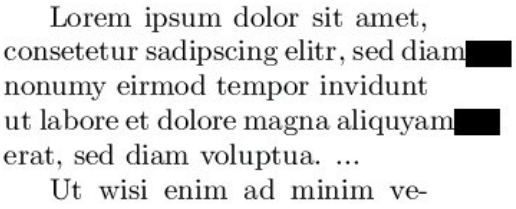
```
\ProvidesPackage{democonv}[Demo article]

\usepackage[paper=a5paper,textwidth=5cm]{geometry}
\setlength\overfullrule{15pt}

\newenvironment{para}{\ignorespaces}{\par}
```

First comes the service header, then, for illustrative purposes, definition of page size and narrow text width. The text will not fit, and TEX will complain and show a dick marker near such places.

Paragraphs in TEX are usually separated by empty lines. But my experience shows that in case of automatic conversion, it is convenient to wrap the paragraphs in environments.

3.2 Tuning the Layout, Remembering the Changes

Compiling the TEX file gives two warning messages about overfull lines, the corresponding places are marked with black squares:



The forward search and reverse search feature [6] of TEX tools helps for big documents. If the feature is enabled, one can jump from a location in the source TEX file to the corresponding place in the output (forward search). And reverse, one can jump from a location in the output to the corresponding place in the source.

For this simple example, no search is required to find the code that should be tuned. A simple correction is enough: just add `\break` after the words "elitr, sed". Now the file is compiled without warnings.

After the layout is corrected, generate a patch file `article.patch` from the original TEX code `article.tex.orig` and the final version `article.tex`:

```
diff -u article.tex.orig article.tex > article.patch
```

The patch file looks as follows:

```
--- article.tex.orig    2010-03-10 04:53:46 +0100
+++ article.tex 2010-03-10 04:56:24 +0100
@@ -5,7 +5,7 @@
 \title{De finibus bonorum et malorum}
 \begin{para}
 Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
-sed diam nonumy eirmod tempor invidunt ut labore et dolore
+sed\break diam nonumy eirmod tempor invidunt ut labore et dolore
 magna aliquyam erat, sed diam voluptua. ...
 \end{para}
 \begin{para}
```

3.3 Publishing an Updated Version

Update the sample XML by adding a paragraph:

```
...
magna aliquyam erat, sed diam voluptua. ...</para>
<para>Duis autem vel eum iriure dolor in hendrerit in vulputate
velit esse molestie consequat, vel illum dolore eu feugiat nulla
facilisis at vero eros et accumsan et ...</para>
<para>Ut wisi enim ad minim veniam, quis nostrud exerci tation
...
```

The output contains the same layout problem as in the original version. The user should repeat the changes. This can be done automatically by applying the patch:

```
patch article2.tex <article.patch;
```

In good cases, like this example, all the changes are automatically applied. But it is possible that some changes are rejected. In this case, the user has to revise the rejections manually. On the bright side, even if something can not be applied automatically, it works as a reminder where to check the layout.

4 Conclusions and Future Work

This paper proposes a new XML-to-paper workflow. The main contribution is emphasis on manual intervention in the generation process. Unlike existing approaches, the workflow helps the user to create and manage layout changes. As a technical implementation, we convert XML to human-friendly TEX code using XSLT and TEXML, and control the changes using diff and patch tools.

The author successfully uses the workflow in internal publishing services in industry. To support the workflow, a helper software named Consodoc (CONStructor Of DOCumentation, `http://getfo.org/consodoc/`) was developed and released to public. But now the software requires a major revision to allow simple creation of processing steps in additional to the default ones.

A part of the workflow, the TEXML-to-TEX converter is an open-source tool (MIT/X Consortium license) written in Python. The software and documentation are available from the home page of TEXML project `http://getfo.org/texml/`, a number of code repositories includes the tool. The private reports from the independent developers indicate that the tool works well in theirs projects.

The future work on the project consist of:

- development of sample TEXML stylesheets for popular XML formats, for example, for DocBook [7];
- revision of Consodoc, development of sample publishing projects, for example, for the book “DocBook 5.0: The Definitive Guide” [8];
- further improvement of the TEXML tool.

References

1. W3C: Extensible Stylesheet Language (XSL), Version 1.0. W3C Recommendation 15 October 2001. http://www.w3.org/TR/2001/REC-xsl-20011015/
2. Knuth, D.: TEX and METAFONT: New Directions in Typesetting, American Mathematical Society and Digital Press, Bedford, MA, 1979.
3. Johnson, M.: Diff, Patch, and Friends. Linux Journal, Volume 1996, Issue 28es (August 1996)
4. Pepping, S.: From XML to TEX, a short overview of available methods, EuroTEX 2001. http://www.ntg.nl/eurotex/PeppingXML.pdf
5. Parashchenko, O: TeXML: Resurrecting TeX in the XML world, TUGboat 28:1, 2007.
6. Laurens, J.: Direct and reverse synchronization with SyncTEX, TUGboat 29:3, 2008.
7. Walsh, N., Muellner, L.: DocBook: The Definitive Guide, O'Reilly & Associates, 1999.
8. Walsh, N.: DocBook 5.0: The Definitive Guide, work in progress, http://www.docbook.org/tdg5/.