TEI-XML to LaTeX workflow: Issues and lessons

Nicolás Vaughan

Abstract

In this paper I discuss some of the issues surrounding the workflow used in the production of the annotated Spanish translation of the medieval work, *Salomon et Marcolfus*. I explain the decisions taken regarding the XSLT transformation of the TEI-XML document, in order to produce a final print-ready PDF from the LuaLaTeX text.

1 Introduction

Several methods are available for converting XML documents into PDF format. Perhaps the most well-known is that of XSL Formatting Objects (XSL-FO) [2]. However, given that XSL-FO was originally designed to produce technical documents, it is quite difficult — if not impossible — to produce other kinds of more sophisticated publications with it. For example, while technically possible, it is impractical to create critical editions using XSL-FO. Moreover, the only available open source application for generating PDFs from XSL-FO sources — the Apache Formatting Objects Processor (FOP) — offers limited compliance with the W3C XSL-FO 1.1 Standard. For example, it does not support interesting features such as indexes and table properties [1], which are now covered by the standard.

Another, more recent, method of producing PDF from XML involves the use of a technology called CSS Paged Media, originally conceived by researchers at Mozilla [3]. Since XML cannot make direct use of CSS, this method requires that the XML sources be transformed — e.g., via XSLT — into HTML files, which, together with the appropriate CSS, can later be converted into PDF. An application is ultimately responsible for carrying out that conversion. Several free and/or open source (FOSS) applications such as wkhtmltopdf [20] and WeasyPrint [16] are available. As in the case of XSL-FO, however, the problem with such FOSS applications is that they provide subpar performance when compared with commercial ones. On the other hand, commercial applications such as PDFreactor [5] and PrinceXML [6] are expensive, often costing thousands of dollars.

Bearing this in mind, the approach we took for the present project was to use XSLT transformations to convert the original XML (more exactly, TEI-XML) into a LuaLaTeX document, which in turn would be used to generate a PDF file. This should allow for the highest possible quality in a print-ready PDF file, while at the same time affording the availability of useful LaTeX packages for generating bibliographies, indexes, nomenclatures, and the like.

Nonetheless, this workflow proved to be not as straightforward as originally thought. Leaving aside the problem of dealing with undesired whitespace (see §4), the greatest complication concerned the decision of where to carry out most of the granular processing (e.g., how to transform an XML `<cit>` element into a full-fledged LaTeX citation) — in the XSLT code or in the LuaLaTeX code. Ideally one wishes for such processing to occur mostly in one or the other of the stages, as this simplifies the code and facilitates debugging. In actuality, however, this was not possible. The idiosyncrasies of each language made it necessary that some parts of the processing be preformed in the first stage, while others in the second one.

2 Background

The project alluded to above is an original annotated translation into Spanish of the Late-Medieval, anonymous work, *Salomon et Marcolfus*, written in Medieval Latin. The objective was to produce a TEI edition of the Latin text (based upon the 1912 critical edition), with an accompanying, digitally-born Spanish translation, also in TEI. An accompanying Middle-English translation, as well as an original critical edition created from the collation of a score of surviving incunabula, are also planned.

All TEI texts were transformed using XSLT into XHTML documents destined for a web version, on the one hand; and into LuaLaTeX documents, which were thereafter compiled into print-ready PDF files destined for digital consumption or for print, on the other hand. The LuaLaTeX version contains indexes of mentioned names and of cited works, manuscripts, and incunabula, as well as a bibliography. It is still a work in progress, and this is the reason why it is not fully available to the public. When ready, it will be made available under the CC BY-NC-SA 4.0 license.

3 Workflow

We first designed a series of XSLT templates [15] corresponding to each TEI-XML element in the source documents.[1] In what follows we will not describe all transformations (the reader can skim the code), but

[1] The precise specification of the TEI documents was inspired by the guidelines described by the Scholastic Commentaries and Texts Archive (SCTA) for diplomatic transcriptions [18, 19]. Since the present document was an annotated translation of a Medieval Latin text, several modifications had to be made. Accordingly, we created ODD [13] and RELAX NG [14] schemas to validate the TEI code.

will refer only to some of the more interesting and challenging.

The templates were used to transform the TEI source documents into LuaLaTeX documents. It is worth mentioning that the decision to use LuaLaTeX rather than PDFLaTeX responded to the need to render Ancient Greek and Hebrew scripts, something not easily possible in the latter. Finally, we used the Java edition of Saxon-HE v.10 [7] to process the XSLT transformations to render LuaLaTeX code, which was thereafter compiled into PDF files.

The main template contains the core of the LuaLaTeX document:

```
<xsl:template match="/">
  \def\SMVersion{<xsl:value-of
    select="$combinedversionnumber"/>}
  \input{smpreamble.tex}
  \begin{document}
    \input{frontmatter.tex}
    \mainmatter
    \pagestyle{smtrad}
    \chapterstyle{smtrad}
    <xsl:apply-templates select="//body"/>
    \input{backmatter.tex}
  \end{document}
</xsl:template>
```

Listing 1: Main XSLT template

To keep it simple, we load the preamble and all macros as external documents: `smmacros.tex`, which in turn is included in `smpreamble.tex`. The whole document is built using the `memoir` class [17], with a few additional packages. One such is the `enotez` package [4], needed to customize the endnotes.

The TEI document contains several levels of logical division, determined by nested `<div>` elements:

```
<body>
  <div> <!--part 1-->
    <div> <!--chapter 1-->
      <p>...</p> <!--paragraph 1-->
      ...
    </div>
    ...
  </div>
  ...
</body>
```

Listing 2: Logical divisions in a TEI document

In XML the fact that one `<div>` element is the child of another `<div>` element is sufficient to establish their logical relationship. In LaTeX, by contrast, logical divisions are determined by the different partitioning commands: `\part`, `\chapter`, `\section`, and so on.

Nevertheless, to facilitate the transformation, we decided to make use of the TEI attribute `@ana` in the `<div>` elements. Accordingly, a `<div>` with `@ana="level1"` is transformed into a LaTeX `\part`; one with `@ana="level2"`, into a LaTeX `\chapter`; and one with no `@ana` into a `\section`. Hence, we created the following XSLT template to transform parts and chapters accordingly:

```
<xsl:template match="div[child::head]">              1
  <xsl:text>&#10;&#10;</xsl:text>                    2
  <xsl:choose>                                       3
    <xsl:when test="head[@ana='level1']">            4
      <xsl:text>\part</xsl:text>                     5
    </xsl:when>                                      6
    <xsl:when test="head[@ana='level2']">            7
      <xsl:text>\chapter</xsl:text>                  8
    </xsl:when>                                      9
    <xsl:otherwise>                                 10
      <xsl:text>\section</xsl:text>                 11
    </xsl:otherwise>                                12
  </xsl:choose>                                     13
  <xsl:text>{</xsl:text>                            14
  <xsl:copy-of select="head"/>                      15
  <xsl:text>}</xsl:text>                            16
  <xsl:if test="@xml:id">                           17
    <xsl:text>\label{</xsl:text>                    18
    <xsl:value-of select="@xml:id"/>                19
    <xsl:text>}</xsl:text>                          20
  </xsl:if>                                         21
  <xsl:text>&#10;&#10;</xsl:text>                   22
  <xsl:apply-templates/>                            23
</xsl:template>                                     24
```

Listing 3: `<div>` sectioning template

Line 15 copies the content of the `<head>` element which is a child of the `<div>`, and uses it as a title for the division. In addition, lines 2 and 22 introduce two line feed characters (Unicode U+0010) in the resulting LaTeX document. This is required both for readability and to create a LaTeX line break. In other templates (e.g., that in Listing 4 below) we simply left a blank line, mainly for readability of the XSLT code.

Paragraph elements (`<p>` in our TEI document) are transformed into LaTeX paragraphs seamlessly, always taking care to leave a line break before and after them. Some paragraphs, however, are treated differently. For some parts of the text in the appendixes it was necessary to create pseudo-headers which were horizontally centred. These were encoded using the TEI attributes `@ana="h1"` and `@ana="h2"` for a `<p>` element. In addition, some non-indented paragraphs were also needed, which were encoded using the attribute `@rend="indented"`.[2] The complete template for `<p>` elements is the following:

```
 1 <xsl:template match="p">
 2   <xsl:if test="@xml:id">
 3     <xsl:text>\label{</xsl:text>
 4     <xsl:value-of select="@xml:id"/>
 5     <xsl:text>}</xsl:text>
 6   </xsl:if>
 7   <xsl:choose>
 8     <xsl:when test="@ana='h1' or @ana='h2'">
 9       <xsl:text>
10         \begin{adjustwidth}{.2\textwidth}%
11           {.2\textwidth}
12           \begin{center}</xsl:text>
13             <xsl:if test="@ana='h1'">
14             <xsl:text>\large{}</xsl:text>
15             </xsl:if>
16             <xsl:apply-templates/>
17             <xsl:text>\end{center}
18         \end{adjustwidth}
19
20         \bigskip
21       </xsl:text>
22     </xsl:when>
23     <!--indented paragraph-->
24     <xsl:when test="@rend='indented'">
25       <xsl:text>\begin{indentedpar}</xsl:text>
26         <xsl:apply-templates/>
27         <xsl:text>\end{indentedpar}</xsl:text>
28     </xsl:when>
29     <!-- -->
30     <xsl:otherwise>
31       <xsl:apply-templates/>
32     </xsl:otherwise>
33   </xsl:choose>
34 </xsl:template>
```

Listing 4: `<p>` template

As can be seen in lines 24–28, a TEI indented paragraph is transformed into a LaTeX `indentedpar` environment, defined like this in the `smmacros.tex` file:

```
\NewDocumentEnvironment{indentedpar}{}
{\begin{list}{}%
   {\setlength\rightmargin{28pt}%
     \setlength\leftmargin{28pt}}%
 \item[]\small\ignorespaces}
 {\end{list}}
```

Listing 5: `indentedpar` environment

[2] According to the TEI P5 Guidelines [11], `@ana` is used to provide an analysis of the element containing it, whereas `@rend` describes the way it is actually rendered.

This illustrates the issue mentioned earlier — where should the piecemeal processing occur. In this case, both the XSLT and the LaTeX code perform part of the processing necessary for producing and typesetting indented paragraphs. Such an ugly "language promiscuity" — so to speak — is exacerbated in the case of citations, quotes, and references. This is mainly because we found it necessary, in the TEI document, to distinguish quotes in several languages: Latin, Spanish, English, Middle English, Anglo Saxon, French, Middle French, Old French, German, Classical Greek, Welsh, Italian, and Hebrew. The XML attribute `@xml:lang`, common to all elements, makes this seemingly easy, for instance:

```
<q xml:lang="fra">
  Ceci n'est pas une pipe.
</q>
```

We follow the ISO 639-3 [8] standard to encode language names: `fra` for current French, `fro` for Old French, `enm` for Middle English, and so on. The problem is that LaTeX's `babel` package uses other language names.[3] So the transformation could not proceed as easily as merely taking the value of `@xml:lang` and using it as an argument to `babel`'s `otherlanguage*` environment or commands to that effect. First we had to translate the ISO codes into `babel` codes using a special function in XSLT:

```
<xsl:template name="my:lang">
  <xsl:param name="lname"/>
  <xsl:choose>
    <xsl:when test="$lname='ang'">english</xsl:when>
    <xsl:when test="$lname='enm'">english</xsl:when>
    <xsl:when test="$lname='eng'">english</xsl:when>
    <xsl:when test="$lname='lat'">latin</xsl:when>
    <xsl:when test="$lname='es'">spanish</xsl:when>
    <xsl:when test="$lname='fro'">french</xsl:when>
    <xsl:when test="$lname='frm'">french</xsl:when>
    <xsl:when test="$lname='fra'">french</xsl:when>
    <xsl:when test="$lname='grc'">greek</xsl:when>
    <xsl:when test="$lname='cym'">welsh</xsl:when>
    <xsl:when test="$lname='deu'">german</xsl:when>
    <xsl:otherwise>english</xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:function name="my:lang" as="xs:string">
  <xsl:param name="lname"/>
  <xsl:call-template name="my:lang">
    <xsl:with-param name="lname" select="$lname"/>
  </xsl:call-template>
</xsl:function>
```

Listing 6: Language code names conversion

[3] After my TUG 2021 presentation, I have been advised — by Frank Mittelbach and others, to all of whom I am grateful — that Babel now supports ISO language codes. This should simplify the XSLT processing here described.

Our original plan was to code this algorithm (and perhaps others) using either pure (La)TeX commands or Lua functions within a LuaLaTeX document. However, this proved to be more difficult than expected,[4] and thus we defaulted to performing the processing both in XSLT and in LaTeX.

We next created the XSLT template to transform `<foreign>`, `<mentioned>`, and `<gloss>` elements, as well as standalone `<q>` elements:

```
 1 <xsl:template match="foreign | mentioned | gloss
 2     | q[not(parent::cit)]">
 3   <xsl:text>\MyQ</xsl:text>
 4   <!--insert lang-->
 5   <xsl:text>{</xsl:text>
 6   <xsl:choose>
 7     <xsl:when test="@xml:lang
 8            or name(.)='foreign'">
 9       <xsl:sequence
10         select="my:lang(@xml:lang)"/>
11     </xsl:when>
12     <xsl:otherwise>
13       <xsl:text>spanish</xsl:text>
14     </xsl:otherwise>
15   </xsl:choose>
16   <xsl:text>}</xsl:text>
17   <!--insert label-->
18   <xsl:text>{</xsl:text>
19   <xsl:if test="@xml:id">
20     <xsl:value-of select="@xml:id"/>
21   </xsl:if>
22   <xsl:text>}</xsl:text>
23   <!--insert text-->
24   <xsl:text>{</xsl:text>
25   <xsl:if test="@ana='lexeme'">
26     <xsl:text>\lexquote{</xsl:text>
27   </xsl:if>
28   <xsl:apply-templates/>
29   <xsl:if test="@ana='lexeme'">
30     <xsl:text>}</xsl:text>
31   </xsl:if>
32   <xsl:text>}</xsl:text>
33 </xsl:template>
```

Listing 7: Language code names conversion

Lines 7–12 call the previous function (Listing 6) and introduce the `babel`-compliant language code. If no language code is specified, the template defaults to Spanish. Line 3 produces a LaTeX command, `\MyQ`, which sorts out the language of the quote, defined as follows:

[4] See the `tex.stackexchange.org` discussions [9, 10].

```
\NewDocumentCommand{\MyQ}{m m +m}
{%
  % check for label
  \ifstrempty{#2}{\relax}{\label{#2}}%
  % check for language
  {%
    \ifstrempty{#1}%
    {\begin{otherlanguage*}{spanish}}%
      % else
      {\begin{otherlanguage*}{#1}}%
        % if spanish
        \ifstrequal{#1}{spanish}%
        % then
        {\enquote{#3}}%
        % else
        {%
          \ifstrequal{#1}{greek}%
          {#3}%
          {\textit{#3}}%
        }%
      \end{otherlanguage*}%
    }%
  }
```

Listing 8: LaTeX code for handling quotations

Handling of citations (which comprise a quote in the original language, an optional quote of the Spanish translation, and a bibliographic reference) is performed in a similar fashion by three different XSLT templates and a series of LaTeX commands.

More interesting, however, is the code used to handle cross-references, both internal and external. We decided to take advantage of the `@type` attribute of TEI's `<ref>` element, which allows for twelve kinds of references, to wit:

Value	**Meaning**
`a`	`\citeauthor`
`p`	`\parencite`
`t`	`\citetitle`
`y`	`\citeyear`
`py`	`\citeyear` in parenthesis
`abbr`	`\citeabbr`
`pabbr`	`\citeabbr` in parenthesis
`abbrpc`	`\citeabbr` (content)
`fulltext`	use `text/rend` without calling `\cite`
`nc`	`\nocite`
`pnc`	`\nocite` in parenthesis
`url`	`\href`

Table 1: Values for `//ref/@type`

For instance, to cite a work from the bibliography using a predefined abbreviation and putting the "content" of the citation inside parentheses (in the `@rend` attribute), the TEI code will be the following:

```
<cit>
  <ref target="#DMLBS" type="abbrpc"
       rend="s.v. 2"/>
</cit>,
```

Listing 9: Example of citation with reference

Some of the twelve possible references are handled by different XSLT templates. For the case chosen above (`@type="abbrpc"`), the code is as follows:

```
<xsl:template match="ref[@type='abbrpc']"
              priority="2">
  <xsl:text>\citeabbr{</xsl:text>
  <xsl:value-of select="my:cleanref(@target)"/>
  <xsl:text>}</xsl:text>
  <!--insert rend/text-->
  <xsl:text>\space(</xsl:text>
  <xsl:choose>
    <!--select @rend if present-->
    <xsl:when test="@rend">
      <xsl:value-of select="@rend"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates/>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:text>)</xsl:text>
</xsl:template>
```

Listing 10: Template for `//ref[@type="abbrpc"]`

As can be seen, we have taken full advantage of `biblatex`'s citation commands, so all processing is performed by LaTeX behind the scenes. This feature, as well as automatic indexes and nomenclatures, justifies our choice of (Lua)LaTeX within the TEI → PDF workflow.

4 Whitespace

One of the tougher problems when dealing with XML and related languages concerns whitespace characters. XML always combines multiple whitespace characters, collapsing them into a single space character. This is sometimes useful, but can create undesirable results when processing TEI documents. Take, for instance, this sample TEI code:

```
<p>
  Some text here that will occupy a couple of
  lines in the main body of the final document.
</p>
<note>
  A textual note for the previous paragraph.
</note>
```

Listing 11: Example of TEI whitespace

Here, indentation and line breaks are needed only for human readability, not for TEI-XML validity. However, since our TEI document was digitally born — i.e., not converted from other sources — and all editorial work is done directly with it, it is imperative that the authors, editors, copy-editors, and other people working with the text can work with it with ease. In short, we did not want to sacrifice code legibility.

The consequent problem is that without further processing and cleaning, the former code will be rendered with an additional space (marked here as a visible space) between the paragraph and the endnote anchor:

…in the main body text of the finished document.␣[1]

Figure 1: Example of rendered text

For typographical reasons, such insertion of whitespace characters is highly undesirable. XSLT can deal with some of these issues, but only with limited success. We used the following code:

```
<xsl:template match="text()">
  <xsl:value-of
    select="replace(., '\s+', ' ')"/>
</xsl:template>
<xsl:function name="my:cleantext">
  <xsl:param name="input"/>
  <xsl:variable name="step1"
    select="replace($input, '\n+', ' ')"/>
  <xsl:variable name="step2"
    select="normalize-space($step1)"/>
  <xsl:sequence select="$step2"/>
</xsl:function>
```

Listing 12: XSLT code for dealing with whitespace

Nonetheless, many spaces still remained in the transformation. Thus, an additional processing step had to be introduced in our workflow, this time in Python [12]. The script we devised is a collection of regex search-and-replace commands such as the following:[5]

```
# Remove trailing whitespace at closing paren
(r" +\)", r")"),
# Remove redundant space before \endnote.
(r"\s*(\\endnote)", r"%\n\n\1"),
# Break line after colon (but skip citations)
(r": +(?!\d)", r":\n"),
```

Listing 13: Python code for dealing with whitespace

[5] More exactly, these are tuples in a list which is then processed with a `re.sub(cpattern, replacement, buffer)` function.

After running the script, unnecessary whitespace is successfully pruned from the output LaTeX document.

5 Conclusions

In the end, the workflow consists of the following steps:

TEI-XML → XSLT → Python → LuaLaTeX → PDF.

To be sure, some of the additional processing — concerned mainly with the transformation of TEI elements into LaTeX commands — might be realised in Lua (within LuaLaTeX). However, it is unlikely that the whitespace problems can be solved without the help of a regex solution (in Python, sed, etc.), as it concerns the passage from one stage to another in the workflow.

References

[1] Apache FOP Compliance Page. `https://xmlgraphics.apache.org/fop/compliance.html`

[2] A. Berglund, ed. Extensible Stylesheet Language (XSL) Version 1.1. W3C Recommendation 05 December 2006. `https://www.w3.org/TR/xsl11/`

[3] E.J. Etemad, S. Sapin, eds. CSS Paged Media Module Level 3. W3C Working Draft, 18 October 2018. `https://www.w3.org/TR/css-page-3/`

[4] C. Niederberger. Enotez—Support for end-notes. `https://ctan.org/pkg/enotez`

[5] PDFreactor. `https://www.pdfreactor.com/`

[6] PrinceXML. `https://www.princexml.com/`

[7] Saxon XSLT and XQuery Processor. `https://sourceforge.net/projects/saxon/files/`

[8] SIL. ISO 639-3. `https://iso639-3.sil.org/`

[9] StackExchange - Problem using macros in directua. `https://tex.stackexchange.com/questions/556911/`

[10] StackExchange - problem with command expansion. `https://tex.stackexchange.com/questions/557079/`

[11] TEI Consortium. TEI P5: Guidelines for Electronic Text Encoding and Interchange. `https://tei-c.org/release/doc/tei-p5-doc/en/html`

[12] N. Vaughan. Cleantex - Python script to remove unwanted whitespace from LaTeX files produced from TEI. `https://github.com/nivaca/cleantex`

[13] N. Vaughan. SM-ODD. ODD schema for the Salomon & Marcolfus project. `https://github.com/nivaca/sm-odd`

[14] N. Vaughan. SM-RELAXNG schema for the Salomon & Marcolfus project. `https://github.com/nivaca/sm-relaxng`

[15] N. Vaughan. XSLT-LaTeX - templates to transform TEI-XML documents into LaTeX. `https://github.com/nivaca/xslt-latex`

[16] WeasyPrint. `https://weasyprint.org/`

[17] P.R. Wilson, L. Madsen. Memoir – Typeset fiction, non-fiction and mathematical books. `https://ctan.org/pkg/memoir`

[18] J.C. Witt, M. Stenskjær, N. Vaughan. Lombard Press Schema 1.0.0 - Diplomatic Transcription Guidelines. `https://community.scta.info/pages/lombardpress-schema-diplomatic.html`

[19] J.C. Witt, M. Stenskjær, N. Vaughan. LombardPress Schema. `https://github.com/lombardpress/lombardpress-schema`

[20] Wkhtmltopdf. `https://wkhtmltopdf.org`

◇ Nicolás Vaughan
Departamento de Literatura
Universidad de los Andes
Bogotá, Colombia
`n.vaughan (at) uniandes.edu.co`
`http://nicolasvaughan.org`
ORCID 0000-0002-2877-0539