

TEI-XML to L^AT_EX 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 LuaL^AT_EX 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) [3]. However, given that XSL-FO was originally designed to produce technical documents, it is quite difficult — if not impossible — to produce other kind of more sophisticated publications with it. For example, while technically possible, it is actually impractical to create critical edition 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 format.

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 [2]. Now 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 [11] and WeasyPrint [10] 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 [6] and PrinceXML [7] are pretty 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 LuaL^AT_EX 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 L^AT_EX 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 (on which 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 L^AT_EX citation) — in the XSLT code or in the LuaL^AT_EX code. Ideally one wishes for such a processing to occur mostly in either one 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 LuaL^AT_EX documents, which were thereafter compiled into print-ready PDF files destined for digital consumption or for print, on the other hand. The LuaL^AT_EX 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 due, it will be made available under the CC BY-NC-SA 4.0 license.

3 The Workflow

We first designed a series of XSLT templates [17] 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 in the guidelines described by the Scholastic Commentaries and Texts Archive (SCTA) for diplomatic transcriptions [19, 20]. Since the present document was an annotated translation of a Medieval Latin text, several modifications had to be made. Accordingly, we created ODD [15] and RELAX NG [16] schemas to validate the TEI code.

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

The templates were used to transform the TEI source documents into Lua \LaTeX documents. It is worth mentioning that the decision to use Lua \LaTeX rather than PDF \LaTeX responded to the need to render Ancient Greek and Hebrew scripts, something not possible in the latter. Finally, we used the Java edition of Saxon-HE v.10 [9] to process the XSLT transformations to render Lua \LaTeX code, which was thereafter compiled into PDF files.

The main template contains the core of the Lua \LaTeX document:

```

1 <xsl:template match="/">
2   \def\SMVersion{<xsl:value-of
3     select="$combinedversionnumber"/>}
4   \input{smpreamble.tex}
5   \begin{document}
6     \input{frontmatter.tex}
7     \mainmatter
8     \pagestyle{smtrad}
9     \chapterstyle{smtrad}
10    <xsl:apply-templates select="//body"/>
11    \input{backmatter.tex}
12  \end{document}
13 </xsl:template>

```

Listing 1: Main XSLT template

To keep it simple, we decided to 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 [18], with very few additional packages. One of them is the `enotez` package [13], needed to customize the endnotes.

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

```

1 <body>
2   <div> <!--part 1-->
3     <div> <!--chapter 1-->
4       <p>...</p> <!--paragraph 1-->
5       ...
6     </div>
7     ...
8   </div>
9   ...
10 </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.

However, in order to facilitate the transformation, we decided make use of a TEI attribute, `@ana`

in the `<div>` elements. Accordingly, a `<div>` with `@ana="level1"` will be 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]">
  <xsl:text>&#10;&#10;</xsl:text>
  <xsl:choose>
    <xsl:when test="head[@ana='level1']">
      <xsl:text>\part</xsl:text>
    </xsl:when>
    <xsl:when test="head[@ana='level2']">
      <xsl:text>\chapter</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>\section</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:text></xsl:text>
  <xsl:copy-of select="head"/>
  <xsl:text></xsl:text>
  <xsl:if test="@xml:id">
    <xsl:text>\label{</xsl:text>
    <xsl:value-of select="@xml:id"/>
    <xsl:text>}</xsl:text>
  </xsl:if>
  <xsl:text>&#10;&#10;</xsl:text>
  <xsl:apply-templates/>
</xsl:template>

```

Listing 3: `<div>` 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 a couple of Line Feed characters (Unicode `
`;) 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 of leaving a line break before and after them. Some paragraphs, however, were 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:

² According to the TEI P5 Guidelines [12], `@ana` is used to provide an analysis of the element containing it, whereas `@rend` describes the way it is actually rendered.

```

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 L^AT_EX `indentedpar` environment, defined like so in the `smmacros.tex` file:

```

1 \NewDocumentEnvironment{indentedpar}{}
2 {\begin{list}{}%
3   {\setlength\rightmargin{28pt}%
4   \setlength\leftmargin{28pt}}%
5   \item[]\small\ignorespaces}
6 {\end{list}}

```

Listing 5: `indentedpar` environment

This illustrates the issue we alluded to before — where should the piecemeal processing occur. In this case, both the XSLT and the L^AT_EX codes perform part of the processing necessary for producing and typesetting indented paragraphs. Such an ugly “language promiscuity” — so to speak — gets exacerbated in the case of citations, quotes, and references. This was 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 pretty easy, for instance:

```

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

```

We follow the ISO 639-3 [4] standard to encode language names: `fra` for current French, `fro` for Old French, `enm` for Middle English, and so on. The problem is that L^AT_EX’s `babel` package uses other language names. So the transformation could not proceed as easily as merely taking the value of `@xml:lang` and use 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='cyn'">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

Our original plan was to code this algorithm (and perhaps others) using either pure (L^A)T_EX commands or Lua functions within a LuaL^AT_EX document. However, this proved to be more difficult than expected,³ and thus we defaulted to performing the processing both in XSLT and in L^AT_EX.

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

³ See the `tex.stackexchange.org` discussions [5, 8] for more details.

```

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 or name(.)='foreign'">
8       <xsl:sequence select="my:lang(@xml:lang)"/>
9     </xsl:when>
10    <xsl:otherwise>
11      <xsl:text>spanish</xsl:text>
12    </xsl:otherwise>
13  </xsl:choose>
14  <xsl:text>}</xsl:text>
15  <!--insert label-->
16  <xsl:text>{</xsl:text>
17  <xsl:if test="@xml:id">
18    <xsl:value-of select="@xml:id"/>
19  </xsl:if>
20  <xsl:text>}</xsl:text>
21  <!--insert text-->
22  <xsl:text>{</xsl:text>
23  <xsl:if test="@ana='lexeme'">
24    <xsl:text>\lexquote{</xsl:text>
25  </xsl:if>
26  <xsl:apply-templates/>
27  <xsl:if test="@ana='lexeme'">
28    <xsl:text>}</xsl:text>
29  </xsl:if>
30  <xsl:text>}</xsl:text>
31 </xsl:template>

```

Listing 7: Language code names conversion

Lines 7–12 call the previous function (in Listing 6) and introduces the correct, `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:

```

1 \NewDocumentCommand{\MyQ}{m m +m}
2 {%
3   % check for label
4   \ifstrempy{#2}{\relax}{\label{#2}}%
5   % check for language
6   {%
7     \ifstrempy{#1}%
8     {\begin{otherlanguage*}{spanish}}%
9     % else
10    {\begin{otherlanguage*}{#1}}%
11    % if spanish
12    \ifstrequal{#1}{spanish}%
13    % then
14    {\enquote{#3}}%
15    % else
16    {%
17      \ifstrequal{#1}{greek}%
18      {#3}%
19      {\textit{#3}}%
20    }%
21    \end{otherlanguage*}%
22  }%
23 }

```

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 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, in order allow for twelve kinds of reference, to wit:

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

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

For instance, if one wished 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:

```

1 <xsl:template match="ref[@type='abbrpc']" priority="2">
2   <xsl:text>\citeabbr{</xsl:text>
3   <xsl:value-of select="my:cleanref(@target)"/>
4   <xsl:text>}</xsl:text>
5   <!--insert rend/text-->
6   <xsl:text>\space</xsl:text>
7   <xsl:choose>
8     <!--select @rend if present-->
9     <xsl:when test="@rend">
10      <xsl:value-of select="@rend"/>
11    </xsl:when>
12    <xsl:otherwise>
13      <xsl:apply-templates/>
14    </xsl:otherwise>
15  </xsl:choose>
16 <xsl:text></xsl:text>
17 </xsl:template>

```

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

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

4 Whitespace

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

```

1 <p>
2   Some text here that will occupy a couple of lines
3   in the main body text of the finished document.
4 </p>
5 <note>
6   This is a textual note for the previous paragraph.
7 </note>

```

Listing 11: Example of TEI whitespace

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

Now the problem is that without further processing and cleaning, the former code will be rendered with an additional space (marked here in red) between the paragraph and the endnote anchor, like so:

... in the main body text of the finished document.¹

Figure 1: Example of rendered text

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

```

<xsl:template match="text()"
1   <xsl:value-of select="replace(., '\s+', ' ')">
2 </xsl:template>
3 <xsl:function name="my:cleantext">
4   <xsl:param name="input"/>
5   <xsl:variable name="step1" select="replace($input,
6     '\n+', ' ')">
7   <xsl:variable name="step2"
8     select="normalize-space($step1)"/>
9   <xsl:sequence select="$step2"/>
10 </xsl:function>
11

```

Listing 12: XSLT code for dealing with whitespace

Nonetheless, a large amount of spaces was still included in the transformation. Thus, an additional processing step had to be introduced in our workflow, this time in Python [14]. The script we devised is a collection of regex search-and-replace commands such as the following:⁴

```

# Remove trailing whitespace at closing parenthesis
1 (r" +\)", r")"),
2
3 # Remove redundant space before \endnote.
4 (r"\s*(\\endnote)", r"%\n\n\1"),
5
6 # Break line after colon (but skip citations)
7 (r": +(?!\d)", r":\n"),
8

```

Listing 13: Python code for dealing with whitespace

After running the script, the output L^AT_EX document is successfully pruned from unnecessary whitespace.

5 Conclusions

In the end, the workflow consisted in the following steps:

TEI-XML → XSLT → Python → LuaL^AT_EX → PDF. To be sure, some of the additional processing — concerned mainly with the transformation of TEI elements into L^AT_EX commands — might be realised in Lua (within LuaL^AT_EX). 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.

⁴ More exactly, these are tuples in a list which is then processed with a `re.sub(cpattern, replacement, buffer)` function.

References

- [1] Apache (TM) FOP Compliance Page. <https://xmlgraphics.apache.org/fop/compliance.html>
- [2] CSS Paged Media Module Level 3. W3C Working Draft, 18 October 2018. <https://www.w3.org/TR/css-page-3/>
- [3] Extensible Stylesheet Language (XSL) Version 1.1. W3C Recommendation 05 December 2006. <https://www.w3.org/TR/xsl11/>
- [4] ISO 639-3. <https://iso639-3.sil.org/>
- [5] Luatex - Problem using macros in directua - TeX - LaTeX Stack Exchange. <https://tex.stackexchange.com/questions/556911/problem-using-macros-in-directua>
- [6] PDFReactor. <https://www.pdfreactor.com/>
- [7] PrinceXML. <https://www.princexml.com/>
- [8] Problem with command expansion - TeX - LaTeX Stack Exchange. <https://tex.stackexchange.com/questions/557079/problem-with-command-expansion>
- [9] Saxon XSLT and XQuery Processor. <https://sourceforge.net/projects/saxon/files/>
- [10] WeasyPrint. <https://weasyprint.org/>
- [11] Wkhtmltopdf. <https://wkhtmltopdf.org/index.html>
- [12] T. Consortium. TEI P5: Guidelines for Electronic Text Encoding and Interchange. <https://tei-c.org/release/doc/tei-p5-doc/en/html/index.html>
- [13] C. Niederberger. Enotez – Support for end-notes. <https://ctan.org/tex-archive/macros/latex/contrib/enotez>
- [14] N. Vaughan. Cleantex - Python script to remove unwanted whitespace from LaTeX files produced from TEI. <https://github.com/nivaca/cleantex>
- [15] N. Vaughan. SM-ODD. ODD schema for the Salomon & Marcolfus project. <https://github.com/nivaca/sm-odd>
- [16] N. Vaughan. SM-RELAXNG schema for the Salomon & Marcolfus project. <https://github.com/nivaca/sm-relaxng>
- [17] N. Vaughan. Xslt-latex. <https://github.com/nivaca/xslt-latex>
- [18] P.R. Wilson, L. Madsen. Memoir – Typeset fiction, non-fiction and mathematical books. <https://ctan.org/tex-archive/macros/latex/contrib/memoir>
- [19] J.C. Witt, M. Stenskjær, N. Vaughan. Lombard Press Schema 1.0.0 - Diplomatic Transcription Guidelines. <http://community.scta.info/pages/lombardpress-schema-diplomatic.html>
- [20] J.C. Witt, M. Stenskjær, N. Vaughan. LombardPress Schema. <https://github.com/lombardpress/lombardpress-schema>

◇ 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