Processing "Computed" Texts

Abstract
This article is a comparison of methods that may be used to derive texts to be typeset by a word processor. By 'derive', we mean that such texts are extracted from a larger structure, which can be viewed as a database. The present standard for such a structure uses an XML-like format, and we give an overview of the available tools for this derivation task.

Keywords
Typesetting computed texts, TEX, LATEX, ConTEXt, XETEX, LuaTEX, XML, XSLT, character maps, XQuery, XSL-FO.

Introduction

Formats based on the TEX typesetting engine—e.g., *Plain TEX* [27], or LATEX [30], or ConTEXt [8], or LuaTEX [9]—are known as wonderful tools to get high-quality print outputs. Of course, they have been initially designed to typeset texts directly written by end-users. But other texts may be *generated dynamically*, in the sense that they result from some *computation* applied to more data, in particular, items belonging to databases. A very simple example is given by a bill computed by means of a spreadsheet program like Microsoft Excel: the master file is an .xls or .xlsx file—that is, all information about data is centralised into this file—but we may wish such a bill to be typeset nicely using a word processor comparable with LATEX[1].

We personally experienced a more significant example: at the University of Franche-Comté, we manage the projects proposed to Computer Science students in several degrees, this curriculum being located at Besançon, in the East of France. That is, we collect projects' proposals, control the assignment of student groups to projects. Then, at the semester's end, we organise the projects' oral defences, and rate students from information transmitted by jurys. During projects, information is transmitted to students and projects' supervisors either on the Web, or by means of printed documents. Managing only a list of project specifications and enriching it progressively is insufficient: it is better for oral defences' announcement to be shown with respect to defences' chronological order, and this order is unknown when projects are proposed. Likewise, we may wish to present the grades received by students according to the decreasing order of these grades, or according to students' alphabetical order, other criteria being possible, too. In these cases, we have to perform a sort operation before typesetting the result. These examples are not limitative: other operations related to 'classical' programming may be needed if we are only interested in a subset of the information concerning projects, for example, extracting projects proposed by companies, not by people working at our university.

It is well-known that TEX's language is not very suitable for tasks directly related to programming[2]. A better idea is to use a format suitable for information management, with interface tools serving several purposes. Database formats could be used, but presently, the indisputable standard to model such formats is XML[3], providing a rich toolbox for this kind of task. In particular, this toolbox provides the XPath language [38], this language's expressions allow parts of an XML document to be addressed precisely. But these tools related to XML have advantages and drawbacks: we are going thoroughly into these points in this article, which is a revised, updated, and extended version of [19]. Reading this article requires only basic knowledge about (LA)TEX and the tools related to XML.

A simple example

The examples given in the introduction are 'real' applications and belong to our framework: Microsoft Excel can generate XML files[4], and we personally manage students' projects by means of a master file using XML-like syntax. However, for sake of simplicity, we consider an easier example for the present article, pictured in Figure 1. This XML text—a file ds.xml—describes some items of a series of stories—*Doc Savage*—first published as *pulps* in the 1930's, then republished as pocket books in the 1960's[5]. As it can be noticed in the given example, the original publication order—for pulps—was not followed by the series of pocket books[6]. In addition, some stories were unpublished as pulps (e.g., *The Red Spider*) or retitled when published as pocket books (e.g., *The Deadly Dwarf*, previously entitled *Repel*), in which case, the pulp's title is given as the `pulp` element's contents[7]. More precisely, if a `pulp` element's contents is empty, this means that the pulp's title was the same as

```
<story-list>
  <story>
    <title>The Deadly Dwarf</title>
    <pulp nb="56">Repel</pulp>
    <pocket-book nb="28"/>
  </story>
  <story>
    <title>The Land of Terror</title>
    <pulp nb="2"/>
    <pocket-book nb="8"/>
  </story>
  <story>
    <title>The Lost Oasis</title>
    <pulp nb="7"/>
    <pocket-book nb="6"/>
  </story>
  <story>
    <title>The Man of Bronze</title>
    <pulp nb="1"/>
    <pocket-book nb="1"/>
  </story>
  <story>
    <title>The Red Spider</title>
    <pocket-book nb="95"/>
  </story>
  <story>
    <title>World&apos;s Fair Goblin</title>
    <pulp nb="74"/>
    <pocket-book nb="39"/>
  </story>
</story-list>
```

Figure 1. Master file using XML-like syntax.

the pocket book's. Fig. 2 gives the schema modelling our taxonomy[8], written using XML Schema [42]. Let us recall that this language provides a *datatype library*: for example, '`xsd:string`' for strings, the prefix '`xsl:`' allows us to get access to XML Schema's constructs[9].

Now we propose to search the information given in Fig. 1, extract the items published as pulps, sort them according to the publication order[10]. The title given is the pulp's; if the corresponding pocket book has been retitled, a footnote must give the 'new' title. Of course, we wish the result to be typeset nicely, as LaTeX or ConTeXt is able to do. To be more precise, a good solution processable by *Plain TeX* could look like the source text given in Fig. 3. As mentioned above, a TeX-based solution:

```
\story-list{%
 \story{\title{...}\pulp{...}...}%
 ...%
}
```

could use TeX commands for dealing with the elements `story-list`, `story`, `title`, `pulp`, and `pocket-book`, but would lead to complicated programming.

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="story-list">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="story"
                     maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="title"
                           type="xsd:string"/>
              <xsd:element name="pulp"
                           minOccurs="0">
                <xsd:complexType>
                  <xsd:simpleContent>
                    <xsd:extension
                      base="xsd:string">
                      <xsd:attribute
                        ref="nb"
                        use="required"/>
                    </xsd:extension>
                  </xsd:simpleContent>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="pocket-book">
                <xsd:complexType>
                  <xsd:attribute
                    ref="nb"
                    use="required"/>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:attribute name="nb"
                 type="xsd:positiveInteger"/>
</xsd:schema>
```

Figure 2. Our organisation expressed in XML Schema.

Using tools related to XML

XSLT producing TeX sources

XSLT[11] [41] is the language designed for transformations of XML texts. By 'transformations', we mean that we can build printed documents as well as online documents to be put on the Web from the source file. 'Simple' texts are possible, too. We also can perform some computation from data stored in the original XML file. Let us come to our example, the text of a stylesheet that may be used to get Fig. 3 from Fig. 1 is given *in extenso* in Fig. 4. This stylesheet takes as much advantage as possible of the features introduced by XPath's and XSLT's new version[12] (2.0): for example, we have made precise the types of the used variables, by means of `as` attributes,

```
\item{1} The Man of Bronze
\item{2} The Land of Terror
\item{7} The Lost Oasis
\item{56} Repel\footnote*{Book's title of pulp \#56:
 The Deadly Dwarf}
\item{74} World's Fair Goblin

\end
```

Figure 3. Stories' titles sorted by pulp numbers.

these types being provided by XML Schema's library[13]. Likewise, we have made precise the types of templates' results. As it can be noticed in Fig. 4, these types belong to XML Schema's namespace, whereas XSLT constructs are prefixed by the namespace associated with '`xsl:`'.

Since we are interested in deriving texts processable by (LA)TEX, an important new feature introduced by XSLT 2.0 is the possible use of *character maps* [41, § 20.1]. In particular, they allow TEX's special characters to be replaced by commands producing them:

`#` ⟶ `\#` `\` ⟶ `$\backslash$`[14]

whenever they appear within a text node to be put by XSLT. More precisely, a single character can be replaced by a string, as shown by the character map `some-special-characters` given in Fig. 4. To distinguish an 'actual' backslash character, belonging to a string, and a command's beginning, a solution is to use a character belonging to a private area of Unicode [34] for the latter. For sake of readabiblity, we define this fictive character by means of an entity[15]—`start-command`—[32, p. 48–49]. Introducing this entity leads us to put a dummy `DOCTYPE` tag, since XSLT stylesheets do not refer to a DTD. In other words, specifying these additional characters is a 'trick', but that allows us to process strings extracted from the original XML file systematically. As shown in Fig. 4, the same technique can be used for opening and closing a group: we use fictive characters the character map transforms into braces. The same for a delimited fragment in (LA)TEX's math mode. Another solution could be the direct generation of Unicode texts and the use of a Unicode-compliant TEX-like engine[16], e.g. XETEX [26] or LuaTEX [9].

As abovementioned, XSLT is not limited to texts' generation, the `xsl:output` element's `method` attribute may also be set to `html` or `xhtml`[17] [41, § 20], in which case it allows Web pages to be generated. Likewise, this `method` attribute set to `xml` means that XML texts are to be generated. Using these output methods provides a great advantage: since any XSLT stylesheet is an XML text, an XSLT processor checks that the final document is well-formed, in particular, opening and closing tags must be balanced. The generation of (LA)TEX texts lacks an analogous check: an XSLT processor cannot ensure that opening and closing braces are balanced, as in TEX; likewise, when LaTEX texts are generated, it is impossible to ensure that environments like:

`\begin{document}` … `\end{document}` (1)

are balanced. Since such errors are not detected statically, they just appear when generated texts are processed. Let us notice that such a check would be more difficult to apply to the texts generated for the ConTEXt format, because the opening and closing commands for ConTEXt's environments are '`\start...`' and '`\stop...`', e.g., the equivalent formulation for (1) in LaTEX is:

`\starttext` … `\stoptext`

in ConTEXt. Concerning the delimiters of a command's arguments, we can ensure that opening and closing braces—more precisely, the two character entities `start-group` and `end-group`, before their replacement by braces—are balanced by using only the XSLT function `ntg:make-group` to build a TEX group from a sequence of strings. Let us remark that XSLT functions have been introduced in XSLT 2.0, so there is an additional reason to use this version. Another solution could be the use of an XML dialect whose architecture would reflect TEX's markup. From our point of view, that would complicate the process since an additional step—a translation from this dialect into 'actual' TEX-like syntax—would be performed. In addition, some versions of such dialects have already been proposed—e.g., in [7, § A.1] for LaTEX—but it seems to us that none is actually used.

XSLT producing XSL-FO texts

Since deriving XML texts by means of XSLT provides a better check level about syntax, an alternative idea is to get XSL-FO[18] [37] texts. Such texts use an XML-like syntax that aims to describe high-quality print outputs. As shown in [16], there is some similarity between LaTEX and XSL-FO, the latter providing, of course, more systematic markup[19]. This language is verbose, but it is not devoted to direct use: XSL-FO texts usually result from applying an XSLT stylesheet. The use of several namespaces—usually denoted by the prefixes '`xsl:`' and '`fo:`'—clearly distinguish elements belonging to XSLT and XSL-FO.

This approach's advantage is clear: generated texts are well-formed. However, XSL-FO lacks *document classes*, as in LaTEX. Some elements allow the description of *page models*, but end-users are entirely responsible for this definition. XSL-FO provides much expressive power about placement of *blocks*[20], but is very basic on other points. For example, let us consider footnotes, end-users are responsible of choosing each footnote

```
<!DOCTYPE stylesheet [<!ENTITY start-command "&#xE000;"> <!ENTITY start-group "&#xE001;">
                      <!ENTITY end-group "&#xE002;">]>

<xsl:stylesheet version="2.0" id="pulps-plus" xmlns:maps="http://www.ntg.nl"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                extension-element-prefixes="xsd">

  <xsl:output method="text" encoding="ISO-8859-1" use-character-maps="some-special-characters"/>

  <xsl:strip-space elements="*"/>

  <xsl:character-map name="some-special-characters">
    <xsl:output-character character="#" string="\#"/>
    <xsl:output-character character="%" string="\%"/>
    <xsl:output-character character="$" string="\$"/>
    <xsl:output-character character="&amp;" string="\&"/>       <!--  &. Plain TeX's commands  -->
    <xsl:output-character character="\" string="$\backslash$"/> <!--  '\{' and '\}' are only  -->
    <xsl:output-character character="{" string="$\{$"/>   <!--  usable in math mode  -->
    <xsl:output-character character="}" string="$\}$"/>   <!--  (cf. [27, Exercise 16.12]).  -->
    <xsl:output-character character="~" string="{\char'7E}"/> <!--  Using hexadecimal code.  -->
    <xsl:output-character character="&start-command;" string="\"/>
    <xsl:output-character character="&start-group;" string="{"/>
    <xsl:output-character character="&end-group;" string="}"/>
  </xsl:character-map>

  <xsl:variable name="eol" select="'&#xA;'" as="xsd:string"/> <!--  (End-Of-Line character.)  -->

  <xsl:template match="story-list" as="xsd:string">
    <xsl:variable name="pulps" as="xsd:string+">
      <xsl:apply-templates select="story[pulp]">
        <xsl:sort select="xsd:integer(pulp/@nb)"/>     <!--  Numerical sort.  -->
      </xsl:apply-templates>
    </xsl:variable>
    <xsl:value-of select="$pulps,$eol,'&start-command;end',$eol" separator=""/>
  </xsl:template>

  <xsl:template match="story" as="xsd:string">
    <xsl:variable name="pulp-0" select="pulp" as="element(pulp)"/>
    <xsl:variable name="pulp-nb-0-string" select="xsd:string($pulp-0/@nb)" as="xsd:string"/>
    <xsl:variable name="pulp-title-0" select="data(pulp-0)" as="xsd:string"/>
    <xsl:variable name="title-processed" as="xsd:string">
      <xsl:apply-templates select="title"/>
    </xsl:variable>
    <xsl:value-of select='"&start-command;item",maps:mk-group($pulp-nb-0-string)," ",
                          if ($pulp-title-0) then
                            $pulp-title-0,"&start-command;footnote*",
                            maps:mk-group(("Book&apos;s title of pulp #",$pulp-nb-0-string,": ",
                                           $title-processed)) else
                            $title-processed,
                          $eol'
                  separator=""/>
  </xsl:template>

  <xsl:template match="title" as="xsd:string"><xsl:apply-templates/></xsl:template>

  <xsl:function name="maps:mk-group" as="xsd:string+">
    <xsl:param name="string-seq" as="xsd:string*"/>
    <xsl:sequence select="'&start-group;',$string-seq,'&end-group;'"/>
  </xsl:function>

</xsl:stylesheet>
```

Figure 4. Getting a source text for *Plain TEX* by means of an XSLT stylesheet.

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>...</fo:layout-master-set>
  <fo:page-sequence master-reference="simple-page" font-family="serif" font-size="medium" text-align="left">
    <fo:flow flow-name="xsl-region-body">
      <fo:list-block provisional-distance-between-starts="20mm" provisional-label-separation="3mm">
        ...
        <fo:list-item>
          <fo:list-item-label start-indent="10mm" end-indent="label-end()">
            <fo:block>56</fo:block>
          </fo:list-item-label>
          <fo:list-item-body start-indent="body-start()">
            <fo:block>
              Repel<fo:footnote>
                <fo:inline font-size="x-small" vertical-align="super">*</fo:inline>
                <fo:footnote-body>
                  <fo:block text-align-last="justify"><fo:leader leader-pattern="rule"/></fo:block>
                  <fo:block font-size="xx-small">
                    <fo:inline font-size="xx-small" vertical-align="super">*</fo:inline>Book's title for
                    pulp #56:  The Deadly Dwarf
                  </fo:block>
                </fo:footnote-body>
              </fo:footnote>
            </fo:block>
          </fo:list-item-body>
        </fo:list-item>
        ...
      </fo:list-block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Figure 5. How to put a footnote in XSL-FO (see the equivalent *Plain TEX* source text in Fig. 3).

mark. Fig. 5 provides the result of applying an XSLT stylesheet providing an XSL-FO text for our example. The first child of the `fo:footnote` element gives the footnote reference, the second child is the actual footnote's contents [37, § 6.12.3]. This `fo:footnote` element seems to be low-level in comparison with LaTeX's `\footnote` command[21]. Of course, if you want footnotes to be numbered automatically, XSLT addresses this problem. However, another point seems to us to be more debatable: end-users are responsible for putting a leader separating footnotes from the text body[22] (we show how to proceed in Fig. 5). So some footnotes may be preceded by a leader, some may not. This point may seem anecdotal, but for LaTeX users, some features of XSL-FO can be viewed as low-level and be difficult to handle since they are already programmed in LaTeX classes.

Last but not least, most current XSL-FO processors do not implement the whole of this language, even if they can successfully process most of XSL-FO texts used in practice[23]. So some features may be unusable, whereas an equivalent construct will work in (LA)TeX. XSL-FO has been designed to deal with the whole of Unicode, so it shows how the Unicode bidirectional algorithm [35] is put into action [18], but this point may also be observed with XeTeX.

XQuery producing TEX sources

XQuery [40] is a query language for data stored in XML form, as SQL[24] does for relational data bases[25]. XQuery can be used to search documents and arrange the result, as an XML structure or a simple text (possibly suitable for a TeX-like engine). An XQuery program processing our example in order to get Fig. 3's text is given in Fig. 6. Like XSLT 2.0, XQuery uses XPath 2.0 expressions and the datatype library provided by XML Schema. As we did in XSLT, we systematically put type declarations using the `as` keyword, for sake of clarity and for taking as much advantage as possible of XQuery's type-checker. Such programs, using FLWOR[26] expressions, are more compact than equivalent ones in XSLT.

However, XQuery is suitable only for generating simple texts: advanced features like character maps in XSLT are provided by some XQuery processors, but are not portable. You have to use the `replace` function [39, § 7.6.3] to deal with TeX's special characters:

```
replace($s,"(#|%)","\\$1")
```

substitutes each occurrence of '#' (resp. '%') by '\#'

```
declare namespace maps = "http://www.ntg.nl" ;
declare namespace saxon = "http://saxon.sf.net/" ;
declare namespace xsd = "http://www.w3.org/2001/XMLSchema" ;

declare option saxon:output "omit-xml-declaration=yes" ;

declare variable $eol as xsd:string := "&#xA;" ;
declare variable $filename as xsd:string external ;

declare function maps:mk-group($string-seq as xsd:string*) as xsd:string+ {
  "{",$string-seq,"}"
} ;

if (doc-available($filename)) then
  string-join((for $story-0 as element(story) in doc($filename)/story-list/story[pulp]
               let $pulp-nb-0 as xsd:untypedAtomic := data($story-0/pulp/@nb),
                   $pulp-nb-0-int as xsd:integer := xsd:integer($pulp-nb-0),
                   $pulp-nb-0-string as xsd:string := xsd:string($pulp-nb-0),
                   $pulp-title-0 as xsd:string := xsd:string(data($story-0/pulp)),
                   $story-title-0 as xsd:string := xsd:string(data($story-0/title))
               order by $pulp-nb-0-int
               return ("\item",maps:mk-group($pulp-nb-0-string)," ",
                       if ($pulp-title-0) then
                         $pulp-title-0,"\footnote*",
                         maps:mk-group(("Book's title of pulp \#",$pulp-nb-0-string,": ",
                                        $story-title-0)) else
                         $story-title-0,$eol),
               $eol,"\end",$eol),
              "") else
  ()
```

Figure 6. Getting a source text for *Plain TEX* by means of XQuery.

(resp. '\%') within the string `$s`. Let us come back to implementation-dependent features, a simple example is given in Fig. 6: we declare that the result is not an XML text by a non-portable option, `saxon:output`[27]. Of course, if XQuery is used to generate XML texts, they are well-formed, but no analogous check can be done about texts generated for a TEX-like engine. In other words, XQuery has the same drawback as XSLT.

A curiosity: DSSSL

DSSSL[28] [21] was initially designed as the stylesheet language for displaying SGML[29] texts. DSSSL includes a core expression language that is a side-effect free subset of the Scheme programming language [25]. XML being a subset of SGML, stylesheets written using DSSSL can be applied to XML texts. DSSSL is rarely used now, but the example we cite illustrates how a functional programming language can be suitable for our requirements. Fig. 7 gives a stylesheet that produces a result equivalent to Fig. 3. In fact, end-users do not write (LA)TEX commands when they develop a stylesheet, the jade[30] program can generate TEX-like texts[31]:

```
jade -d pulps.dsl -t tex ds.sgml
```

—we have to specify a predefined *backend*, here '`tex`'— the typesetting engine usable to process `jade`'s results being JadeTEX [7, § 7.5.2]. Deriving texts directly processable by LaTEX or ConTEXt is impossible.

As shown in Fig. 7, processing a *`name`* element uses pattern-matching:

```
(element name E)
```

the *E* expression consists of assembling *literals* by means of the `make` form, using types predefined in DSSSL: `paragraph`, `sequence`, ... The generic type of such results is called *sosofo*[32] w.r.t. DSSSL's terminology.

Enriched TEX engines

If we go back to programs based on TEX-like typesetting engines, there are two other possible methods, based on TEX-engines 'enriched' by using a 'more classical' programming language. In both cases, XML texts are preprocessed by procedures belonging to a programming language, and the result is sent to a TEX-engine. Of course, such a *modus operandi* is suitable only if we want to generate (LA)TEX texts, it would be of little interest to get XML texts or pages written using (X)HTML[33].

PyTEX [6] is written using Python [28] and uses TEX as a daemon. Getting the components of a 'computed'

```
<!DOCTYPE style-sheet PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">

<style-sheet>
  <style-specification id="pulps-plus">
    <style-specification-body>
      (declare-flow-object-class page-footnote
        "UNREGISTERED::Sebastian Rahtz//Flow Object Class::page-footnote")

      (root (let ((margin-size 1in))
              (make simple-page-sequence
                page-width: 210mm page-height: 297mm left-margin: margin-size right-margin: margin-size
                top-margin: margin-size bottom-margin: margin-size header-margin: margin-size
                footer-margin: 12mm center-footer: (page-number-sosofo)
                (process-children))))
      (element story-list-sgml
        (make sequence
          (let ((get-pulp (lambda (node-list) (select-elements (children node-list) "pulp"))))
            (process-node-list
             (apply node-list
                    (list-stable-sort
                     (node-list->list (node-list-filter (lambda (node-list)
                                                          (not (node-list-empty? (get-pulp node-list))))
                                                        (children (current-node))))
                     <
                     (lambda (story-node-list)
                       (string->number (attribute-string "nb" (get-pulp story-node-list))))))))))
      (element story
        (let* ((story-indent 20pt)
               (current-children (children (current-node)))
               (pulp-node-list (select-elements current-children "pulp"))
               (pulp-processed (process-node-list pulp-node-list))
               (pulp-title-string (data pulp-node-list))
               (title-processed (process-node-list (select-elements current-children "title"))))
          (make paragraph
            first-line-start-indent: (- story-indent) font-size: 12pt space-before: 10pt
            start-indent: story-indent pulp-processed space-literal
            (if (string-null? pulp-title-string)
                title-processed
                (let* ((footnote-marker-sosofo (literal "*"))
                       (footnotemark-sosofo (make-superscript footnote-marker-sosofo)))
                  (make sequence
                    (literal pulp-title-string) footnotemark-sosofo
                    (make page-footnote
                      footnotemark-sosofo (literal "Book's title of pulp #") pulp-processed (literal ": ")
                      title-processed)))))))
      (element title (process-children-trim))
      (element pulp (literal (attribute-string "nb")))
      (define space-literal (literal " "))
      (define make-superscript
        (let ((shift-factor 0.4)
              (size-factor 0.6))
          (lambda (sosofo-0)
            (make sequence
              font-size: (* (inherited-font-size) size-factor)
              position-point-shift: (* (inherited-font-size) shift-factor)
              sosofo-0))))

      (define (string-null?  string-0) ...)
      (define (list-stable-sort list-0 rel-2? key-f1)
        ;; Sorts list-0 according to the order relation rel-2?. The argument key-f1 gives a key for each element.
        ...)
    </style-specification-body>
  </style-specification>
</style-sheet>
```

Figure 7. DSSSL stylesheet generating a text to be printed.

text is left to the Python functions dealing with XML texts and successive results are sent to TEX, in turn.

LuaTEX [9] is able to call functions written using Lua [20]. On another point, this TEX-engine can process texts using XML-like syntax, as shown in [10]. Fig. 8 gives an implementation of our example in ConTEXt MkIV: it uses Lua to define an interface with sorting functions, the other functionalities being put into action using TEX-like commands. As in ConTEXt, the layout is controlled by set-up commands:

```
\start...setup  ...  \stop...setup
```

—for example, `title` elements' contents are just displayed, processing `pulp` elements displays the number or the title, depending on a mode—then our XML file is processed 'atomically', by means of the `\xmlprocessfile` command. This approach is promising, but let us recall that LuaTEX and ConTEXt MkIV have not yet reached stable state: that is planned for the year 2012. Another important drawback: as shown by the examples using the `\xmlfilter` command in Fig. 8, this command uses path expressions, very close to XPath expressions, but not identical. For example, `command`—used to connect a selected item to the set-up command that processes it—obviously does not belong to XPath. On the contrary, some XPath expressions are not recognised, even if 'simple' paths are processed. Some tricks may be used as workarounds, but we personally think that complete compatibility with XPath should be attained.

Conclusion

If we sum up the approaches shown throughout this article, those that seem suitable are XQuery for simple examples, XSLT for more ambitious ones, provided that Version 2.0 is used. The use of LuaTEX could be interesting in a near future, too.

However, we think that there are two directions that should be explored. The first would be a modern implementation of XSL-FO using a TEX-like typesetting engine. Such an implementation has begun: Passive TEX [4], but this project is presently stalled. We think that processing XSL-FO could re-use the experience accumulated by (LA)TEX developers, even if syntaxes are very different[34]. The second direction would be the definition and implementation of an output mode of XSLT suitable for (LA)TEX; some additional services could be performed: for example, checking that braces and environments are balanced. Such an output mode already exists in nbst[35], the language of bibliography styles close to XSLT and used by MlBibTEX[36] [11], but it concerns only the way to process LaTEX's special characters. If the `method` attribute of the `nbst:output` element is set to `text`, the result of:

```
\enablemode[ds:pulp]

\startxmlsetups xml:ds:base
 \xmlsetsetup{#1}{%
  story-list|story|title|pulp|pocket-book}{%
  xml:ds:*}
\stopxmlsetups

\xmlregisterdocumentsetup{ds}{xml:ds:base}

\startxmlsetups xml:ds:story-list
 \xmlresetsorter{story}
 \xmlfilter{#1}{%
  /story/command(xml:story-list:getkeys)}
 \subject{sortkeys}
 \xmlshowsorter{story}\blank
 \xmlsortentries{story}
 \xmlflushsorter{story}{xml:story-list:flush}
\stopxmlsetups

\startxmlsetups xml:story-list:getkeys
 \xmladdsortentry{story}{#1}{%
  \xmlattribute{#1}{/pulp}{nb}}
\stopxmlsetups

\startxmlsetups xml:story-list:flush
 \startitemize\xmlfirst{#1}{.}\stopitemize
\stopxmlsetups

\startxmlsetups xml:ds:story
 \sym{\xmlfirst{#1}{pulp}}
 \xmldoifelsetext{#1}{pulp}{
  {\disablemode[ds:pulp] \xmlfirst{#1}{pulp}
   \footnote{%
    Book's title: \xmlfirst{#1}{title}}}{%
  \xmlfirst{#1}{title}}
\stopxmlsetups

\startxmlsetups xml:ds:title
 \xmlflush{#1}
\stopxmlsetups

\startxmlsetups xml:ds:pulp
 \doifmodeelse{ds:pulp}{\xmlatt{#1}{nb}}{%
  \xmlflush{#1}}
\stopxmlsetups

\starttext
 \xmlprocessfile{ds}{ds.xml}{}
\stoptext
```

Figure 8. Processing a master file with ConTEXt Mk IV.

```
<nbst:text>#60 The Maji</nbst:text>
```

is '`#60 The Maji`'. If this attribute is set to `LaTeX`, the result is '`\#60 The Maji`'.

Acknowledgements

I wish to thank Hans Hagen who greatly helped me debug and improve LuaTEX source texts. Thanks also to Karl Berry, who clarified some terminology notions.

Last but not least, thanks to Taco Hoekwater for his patience when he was waiting for this final version.

References

[1] *Apache FOP*. November 2010. `http://xmlgraphics.apache.org/fop/`.

[2] Frédéric Boulanger : « LaTeX au pays des tableurs ». *Cahiers GUTenberg*, Vol. 39–40, p. 7–16. In *Actes du Congrès GUTenberg 2001*, Metz. Mai 2001.

[3] Neil Bradley: *The Concise SGML Companion*. Addison-Wesley. 1997.

[4] David Carlisle, Michel Goossens et Sebastian Rahtz : « De XML à PDF avec xmltex, XSLT et PassiveTeX ». *Cahiers GUTenberg*, Vol. 35–36, p. 79–114. In *Actes du congrès GUTenberg 2000*, Toulouse. Mai 2000.

[5] James Clark *et al.*: *Relax NG*. `http://www.oasis-open.org/committees/relax-ng/`. 2002.

[6] Jonathan Fine: "TeX Forever!". In: *Proc. EuroTeX 2005*, pp. 150–158. Pont-à Mousson, France. March 2005.

[7] Michel Goossens and Sebastian Rahtz, with Eitan M. Gurari, Ross Moore and Robert S. Sutor: *The LaTeX Web Companion*. Addison-Wesley Longman, Inc., Reading, Massachusetts. May 1999.

[8] Hans Hagen: *ConTeXt, the Manual*. November 2001. `http://www.pragma-ade.com/general/manuals/cont-enp.pdf`.

[9] Hans Hagen: "The Luafication of TeX and ConTeXt". In: *Proc. BachoTeX 2008 Conference*, pp. 114–123. April 2008.

[10] Hans Hagen: "Dealing with XML in ConTeXt MkIV". *MAPS*, Vol. 37, pp. 25–39. 2008.

[11] Jean-Michel Hufflen: "MlBibTeX's Version 1.3". *TUGboat*, Vol. 24, no. 2, pp. 249–262. July 2003.

[12] Jean-Michel Hufflen: "Introduction to XSLT". *Biuletyn GUST*, Vol. 22, pp. 64. In *BachoTeX 2005 conference*. April 2005.

[13] Jean-Michel Hufflen: "Advanced Techniques in XSLT". *Biuletyn GUST*, Vol. 23, pp. 69–75. In *BachoTeX 2006 conference*. April 2006.

[14] Jean-Michel Hufflen: "Introducing LaTeX users to XSL-FO". *TUGboat*, Vol. 29, no. 1, pp. 118–124. EuroBachoTeX 2007 proceedings. 2007.

[15] Jean-Michel Hufflen: "XSLT 2.0 vs XSLT 1.0". In: *Proc. BachoTeX 2008 Conference*, pp. 67–77. April 2008.

[16] Jean-Michel Hufflen : « Passer de LaTeX à XSL-FO ». *Cahiers GUTenberg*, Vol. 51, p. 77–99. Octobre 2008.

[17] Jean-Michel Hufflen: "Introduction to XQuery". In: Tomasz Przechlewski, Karl Berry and Jerzy B. Ludwichowski, eds., *TeX: at a Turning Point, or at the Crossroads? Proc. BachoTeX 2009 Conference*, pp. 17–25. April 2009.

[18] Jean-Michel Hufflen: "Multidirectional Typesetting in XSL-FO". In: Tomasz Przechlewski, Karl Berry and Jerzy B. Ludwichowski, eds., *TeX: at a Turning Point, or at the Crossroads? Proc. BachoTeX 2009 Conference*, pp. 37–40. April 2009.

[19] Jean-Michel Hufflen: "Processing 'Computed' Texts". *ArsTeXnica*, Vol. 8, pp. 102–109. In GUIT 2009 meeting. October 2009.

[20] Roberto Ierusalimschy: *Programming in Lua*. 2nd edition. Lua.org. March 2006.

[21] International Standard ISO/IEC 10179:1996(E): *DSSSL*. 1996.

[22] ISO/IEC 19757: *The Schematron. An XML Structure Validation Language Using Patterns in Trees*. `http://www.ascc.net/xml/resource/schematron/schematron.html`. June 2003.

[23] Michael H. Kay: *XSLT 2.0 Programmer's Reference*. 3rd edition. Wiley Publishing, Inc. 2004.

[24] Michael H. Kay: *Saxon. The XSLT and XQuery Processor*. October 2010. `http://saxon.sourceforge.net`.

[25] Richard Kelsey, William D. Clinger, and Jonathan A. Rees, with Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman and Mitchell Wand: "Revised^5 Report on the Algorithmic Language Scheme". *HOSC*, Vol. 11, no. 1, pp. 7–105. August 1998.

[26] Jonathan Kew: "XeTeX in TeX Live and beyond". *TUGboat*, Vol. 29, no. 1, pp. 146–150. EuroBachoTeX 2007 proceedings. 2007.

[27] Donald Ervin Knuth: *Computers & Typesetting. Vol. A: The TeXbook*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.

[28] Alex Martelli: *Python in a Nutshell*. 2nd edition. O'Reilly. July 2006.

[29] Jim Melton and Alan R. Simon: *Understanding the new SQL*. Morgan Kaufmann. 1993.

[30] Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig and Joachim Schrod: *The LaTeX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Mas-

sachusetts. August 2004.
[31] Chuck Musciano and Bill Kennedy: *HTML & XHTML: The Definitive Guide*. 5th edition. O'Reilly & Associates, Inc. August 2002.
[32] Erik T. Ray: *Learning XML*. O'Reilly & Associates, Inc. January 2001.
[33] Denis B. Roegel : « Anatomie d'une macro ». *Cahiers GUTenberg*, Vol. 31, p. 19–27. Décembre 1998.
[34] The Unicode Consortium: *The Unicode Standard Version 5.0*. Addison-Wesley. November 2006.
[35] The Unicode Consortium, `http://unicode.org/reports/tr9/`: *Unicode Bidirectional Algorithm*. Unicode Standard Annex #9. March 2008.
[36] Eric van der Vlist: *Comparing XML Schema Languages*. `http://www.xml.com/pub/a/2001/12/12/schemacompare.html`. December 2001.
[37] W3C: *Extensible Stylesheet Language (XSL). Version 1.1*. W3C Recommendation. Edited by Anders Berglund. December 2006. `http://www.w3.org/TR/2006/REC-xsl11-20061205/`.
[38] W3C: *XML Path Language (XPath) 2.0*. W3C Recommendation Draft. Edited by Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael H. Kay, Jonathan Robie and Jérôme Siméon. January 2007. `http://www.w3.org/TR/2007/WD-xpath20-20070123`.
[39] W3C: *XQuery 1.0 and XPath 2.0 Functions and Operators*. W3C Recommendation. Edited by Ashok Malhotra, Jim Melton, and Norman Walsh. January 2007. `http://www.w3.org/TR/2007/REC-xpath-functions-20070123`.
[40] W3C: *XQuery 1.0: an XML Query Language*. W3C Recommendation. Edited by Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie and Jérôme Siméon. January 2007. `http://www.w3.org/TR/xquery`.
[41] W3C: *XSL Transformations (XSLT). Version 2.0*. W3C Recommendation. Edited by Michael H. Kay. January 2007. `http://www.w3.org/TR/2007/WD-xslt20-20070123`.
[42] W3C: *XML Schema*. December 2008. `http://www.w3.org/XML/Schema`.
[43] Larry Widen and Chris Miracle: *Doc Savage: Arch Enemy of Evil*. Fantasticon Press, Milwaukee, Wisconsin. 1993.

Notes

1. Let us mention that [2] shows—in French—how to use LaTeX to put spreadsheets' functionalities into action.

2. As an example of using TeX's language for programming purposes, readers interested in putting a sort procedure into action can refer to [33]: this *modus operandi* may be viewed as a worthwhile exercise, but is unusable in practice, especially when it is not trivial to obtain sort keys from items to be sorted.

3. e**X**tensible **M**arkup **L**anguage. Readers interested in a general introductory book to this formalism can refer to [32].

4. The XML format used by Microsoft Excel is OOXML (**O**ffice **O**pen XML).

5. All the source texts mentioned throughout [19] and this article—including new versions realised for this present article, in which case the corresponding file names are suffixed with '`-plus`'—can be downloaded *in extenso* from the Web page `http://lifc.univ-fcomte.fr/home/~jmhufflen/texts/guit-2009/`.

6. If you are interested in the story of *Doc Savage* series and its successive editions, you can find more information in [43].

7. When several titles have been used, such a story is more commonly known under the pocket book's title, because pocket books are easier to get than pulps, which are very rare. That is why our `title` elements always refer to pocket books', the contents of `pocket-book` elements being always empty.

8. *Schemas* allow specifiers to define *document types*, which can be viewed as some taxonomy common to a family of XML texts. There exist several schema languages, and the Web page abovementioned gives several versions, using a DTD (**D**ocument **T**ype **D**efinition) [32, Ch. 5], XML Schema [42], Relax NG (**N**ew **G**eneration) [5], and Schematron [22]. A discussed comparison among these schema languages can be found in [36].

9. Readers interested in more details about XML namespaces can consult [32, pp. 41–45].

10. This example seems to us to be pertinent, because there is no order 'better' than others: the original order is based on pulps, but—as mentioned above—some stories are unpublished as pulps, whereas sorting stories according to pocket books' order allows us to sort all the stories, but this is not really chronological.

11. e**X**tensible **S**tylesheet **L**anguage **T**ransformations. Introductions to this language have been given in some BachoTeX conferences, held in Poland: [12, 13, 15].

12. A study of XPath 2.0's and XSLT 2.0's new features, in comparison with XPath 1.0 and XSLT 1.0, can be found in [15].

13. In addition, let us mention that when an XML text is processed by XSLT 2.0, the information put into a DTD or an XML Schema text can be exploited [23, p. 58]. That is not true about other schema languages.

14. In text mode, LaTeX 2_ε's modern versions provide the `\textbackslash` command [30, Table 7.33].

15. If we name this character by introducing a variable by means of an `xsl:variable` element—as we did in Fig. 4 for the end-of-line character—we cannot use this variable's value within the `character` attribute of the `xsl:output-character` element.

16. Engines are not formats: a *format* is a set of pre-loaded definitions based on primitives of a TeX-like engine, whereas an *engine* is TeX or is derived from TeX by adding or redefining

some primitives. *Plain TEX* and LaTEX are formats, XeTEX and LuaTEX are engines.
17. (e**X**tensible) **H**yper**T**ext **M**arkup **L**anguage. XHTML is a reformulation of HTML—the original language of *Web* pages—using XML conventions. [31] is a good introduction to these languages.
18. e**X**tensible **S**tylesheet **L**anguage–**F**ormatting **O**bjects.
19. [16] is written in French. If you would like a similar text in English, [14] is an abridged version.
20. Roughly speaking, a *block* in XSL-FO is analogous to a *minipage* in LaTEX [16, § 1.2].
21. XSL-FO's footnotes can be compared with *Plain TEX*'s `\footnote` command, as shown in Fig. 3.
22. That is done in standard LaTEX class, but not universal: as an example related to TEX's community, the arstexnica class, used for articles of the ArsTEXnica journal—published by GUIT (*Gruppo Utilizzatori Italiani di TEX*), the Italian-speaking TEX users group—does not put leaders just above footnotes.
23. We personally use Apache FOP (**F**ormatting **O**bjects **P**rocessor) [1]:

```
fop pulps-result.fo pulps-result.pdf
```

generates a PDF (*P*ortable **D**ocument **F**ormat) file from a source text in XSL-FO.
24. **S**tructured **Q**uery **L**anguage. A good introductory book about it is [29].
25. A short introduction to XQuery is given in [17].
26. '**F**or, **L**et, **W**here, **O**rder by, **R**eturn', the keywords used throughout such expressions.
27. The XQuery processor we have used for this example is Saxon [24]. Fig. 6's text can be processed by:

```
java net.sf.saxon.Query pulps-plus.xq \
  filename="ds.xml"
```

We also use Saxon as an XSLT 2.0 processor and the stylesheet of Fig. 4 can be processed by:

```
java net.sf.saxon.Transform -s:ds.xml \
  -xsl:pulps-plus.xsl
```

28. **D**ocument **S**tyle **S**emantics **S**pecification **L**anguage.
29. **S**tandard **G**eneralised **M**arkup **L**anguage. Now it is only of a historical interest. Readers interested in this metalanguage can refer to [3].
30. **J**ames Clark's **A**wesome **D**SSSL **E**ngine.
31. We use a different file and a different name for the root element (`story-list-sgml`) because of syntactic reasons: empty tags' syntax was different in SGML [3, p. 259].
32. **S**pecification **O**f a **S**equence **O**f **F**low **O**bjects.
33. Unless a converter to (X)HTML is used, of course.
34. Besides, it is well-known that TEX recognises only its own formats, which complicates cooperation between TEX and other programs.
35. **N**ew **B**ibliography **STy**les.
36. **M**ulti**L**ingual BIBTEX.

Jean-Michel Hufflen
LIFC (EA CNRS 4157),
University of Franche-Comté, 16, route de Gray,
25030 Besançon Cedex, France