Bibliography Styles Easier with MlBIBTEX

Jean-Michel HUFFLEN
LIFC (FRE CNRS 2661)
University of Franche-Comté
16, route de Gray
25030 BESANÇON CEDEX
FRANCE
hufflen@lifc.univ-fcomte.fr
http://lifc.univ-fcomte.fr/~hufflen

Abstract

We emphasise and discuss some methodology about writing bibliography styles using the nbst language, part of MlBIBTEX. Most of the given tricks can also be applied to developing styles using XSLT, since nbst extends it closely. Last we show that the organisation of a bibliography style in several files allows modular decomposition.

Keywords: bibliographies, methodology, bibliography styles, multilingual features, BIBTEX, MlBIBTEX, bst, nbst, XML, XSLT.

Résumé

Nous dégageons et argumentons quelques méthodes d'écriture de styles bibliographiques au moyen du langage nbst de MlBIBTEX. La plupart des conseils donnés peuvent également s'appliquer au développement de styles en XSLT, le langage nbst en étant assez proche. Enfin, nous montrons en quoi l'organisation des divers fichiers d'un style bibliographique permet une décomposition modulaire.

Mots-clés : bibliographies, méthodologie, styles bibliographiques, multilinguisme, BIBTEX, MlBIBTEX, bst, nbst, XML, XSLT.

Zusammenfassung

Es werden einige Methoden dargelegt und untergesucht, um bibliographische Styles in der Sprache nbst zu schreiben. Da nbst mit XSLT nah verwandt ist, kann diese Anleitung auch für die Programmierung der Styles in XSLT helfen. Am Ende wird an der Aufteilung der bibliographischen Styles in einzelne Dateien gezeigt, dass eine modulare Dekomposition möglich ist.

Stichwörter: Bibliographien, Methodik, bibliographischen Styles, mehrsprachigen Funktionen, BIBTEX, MlBIBTEX, bst, nbst, XML, XSLT.

Introduction

This article aims to give some methodology about the development of *bibliography styles*, that is, specifications that rule the layout of *references* put in the 'Bibliography' section of a document, these references being built from *entries* located in bibliography data bases.

When we started the development of our program MlBIBTEX (for '**M**ulti**L**ingual BIBTEX') [9], we were interested in going thoroughly into multilingual aspects for a bibliography processor belonging to the programs of TEX's family and especially, generating bibliographies as source files for the LATEX word processor [22], like BIBTEX [26]. More precisely, we aimed to put into action an 'extended' BIBTEX with multilingual features comparable with LATEX's. Another example of such an extension is given by the babelbib package and the bibliography styles in interface with it [7].

As we explained in [12], we think that such organisation — adopted for MlBIBTEX's first version [9] — leads to complicated bibliography styles, since the language bst [25], used within BIBTEX, is not modular: each style is a monolithic program put in

```
@INPROCEEDINGS{thys1997,
        AUTHOR = {first => Frank,
                  last => Thys},
        TITLE = {Auf der {Spur} des
                 {Vernichters}},
        BOOKTITLE = {Dinoland},
        EDITOR = {first => Wolfgang,
                  last => Holbein},
        PAGES = {353--437},
        PUBLISHER = {Bastei L\"{u}bbe},
        ADDRESS = {Bergisch Gladbach},
        YEAR = 1997,
        MONTH = aug,
        LANGUAGE = german}
```

Figure 1: Entry using MlBibTeX's syntax.

only a single file, so if we would like to add multilingual features, we have to extend each style separately. This point and others decided us to develop a new language, so-called nbst, for '**n**ew **b**ibliography **st**yles', close to XSLT[1], the language of transformations for XML[2] documents. We think that such a choice is good, since XML becomes a central formalism for document interchange. In particular, using nbst eases the production of bibliographies for XML documents: for instance, documents written using XSL-FO[3] [37], a language for describing high-quality print outputs, or DocBook [38], a system for writing structured documents.

We explain in [17] why MlBibTeX does not use XSLT itself, after converting bibliography (.bib) files into an XML-like format, as programs like BibteXML [6] or BIB2XML [27] do. However, if we agree to consider an XSLT-like language for bibliography styles, we have to rewrite most of the bibliography styles of BibTeX, if we want to provide some continuity with this program. There exists a way to import bst functions into an nbst program [11], nevertheless it is obvious that complete rewriting is prefereable, in order to take as much advantage as possible of this programming paradigm. We put some methodology into action to rewrite BibTeX's bibliography styles, we are giving these methods hereafter.

```
<inproceedings id="thys1997" language="german">
  <author>
    <name>
      <personname>
        <first>Frank</first><last>Thys</last>
      </personname>
    </name>
  </author>
  <title>
    Auf der <asitis>Spur</asitis> des
    <asitis>Vernichters</asitis>
  </title>
  <booktitle>Dinoland</booktitle>
  <editor>
    <name>
      <personname>
        <first>Wolfgang</first>
        <last>Holbein</last>
      </personname>
    </name>
  </editor>
  <publisher>Bastei Lübbe</publisher>
  <year>1997</year>
  <month><aug/></month>
  <address>Bergisch Gladbach</address>
  <pages>
    <firstpage>353</firstpage>
    <lastpage>457</lastpage>
  </pages>
</inproceedings>
```

Figure 2: The entry of Figure 1 as an XML tree.

We begin with a small example, in order to illustrate the expressive power of nbst. Second we show how to design the layout of a reference. We consider a particular case: the `@INPROCEEDINGS` entry type of BibTeX — for an article in a conference proceedings or a story in an anthology — but our method is easily adaptable to any entry type. Then we implement our specification. Last, we show how to organise the different items of a bibliography and give some advice about the decomposition of an nbst program into several files. A succint comparison between bst and nbst statements is given as an annexe, followed by some complements about writing external functions using Scheme — the language used for developing MlBibTeX [15] — close to the expression language used as part of DSSSL[4] [18], the language of stylesheets of SGML[5] [8].

What knowledge is required to read this article? A basic one about XML, XPath — the language used to address parts of an XML document — and XSLT is sufficient to just understand the examples given hereafter. Good introductions to them are [29, 30, 34], the 'official' references about XPath and XSLT, issued by the W3C[6], are [36, 35]. Concerning

[1] e**X**tensible **S**tylesheet **L**anguage **T**ransformations.

[2] e**X**tensible **M**arkup **L**anguage.

[3] e**X**tensible **S**tylesheet **L**anguage — **F**ormatting **O**bjects.

[4] **D**ocument **S**tyle **S**emantics and **S**pecification **L**anguage.

[5] **S**tandard **G**eneralized **M**arkup **L**anguage, the ancestor of XML. Now it has just historical interest.

[6] **W**orld **W**ide **W**eb **C**onsortium.

```
<!ELEMENT pages (onepage+ |
                 (firstpage,(ff | lastpage)) |
                 pages-verbatim)>
<!ELEMENT onepage         %INTEGER;>
<!ELEMENT firstpage       %INTEGER;>
<!ELEMENT lastpage        %INTEGER;>
<!ELEMENT ff              EMPTY>
<!ELEMENT pages-verbatim  (#PCDATA)>
<!--  Strictly speaking, '%INTEGER;' is a parameter
      entity (cf. [29, pp. 163–164]) standing for parsed
      character data ('#PCDATA'). But we use it for
      sake of readability, whenever the content of a
      text node is an integer, because DTDs'
      formalism does not know this type. 'ff' is for
      an unspecified number of following pages.
  -->
```

Figure 3: Excerpt from our DTD: specification of pages from a journal or book.

```
FUNCTION {multi.page.check}
{ 't :=    % t is given the value of the PAGES field,
           % popped from the stack.
  #0 'multiresult :=    % I.e., multiresult ← false.
    { multiresult not   % While multiresult is
      t empty$ not      % false and t non-empty,
      and               % do
    }
    { t #1 #1 substring$        % compare t's first
      duplicate$ "-" =          % character with
      swap$ duplicate$ "," =    % '-', ',', '+';
      swap$ "+" =
      or or
        % if success, update multiresult;
        { #1 'multiresult := }
        % if not, update t by removing its head:
        { t #2 global.max$ substring$ 't := }
      if$
    }
  while$
  multiresult   % pushed result.
}
```

Figure 4: How BibTEX detects that several page numbers are given.

MlBibTEX more precisely, all its elements and functions used within path expressions are described in [13]. On another point, we think that developing new functions in Scheme by MlBibTEX's end-users is only needed for very specific applications, so referring to an introductory book such as [32] is sufficient to understand the given examples. MlBibTEX has been developed using the fifth revision of this language [19].

A small example

Let us consider the bibliographical entry given in Figure 1. Even if it roughly looks like a BibTEX entry, we can notice the use of syntactic features specific to MlBibTEX: a `LANGUAGE` field[7], some keywords for introducing the different parts of a person name: '`first`', '`last`'. All these syntactic features are described precisely in [13].

If this entry is cited throughout a document, the corresponding bibliographical reference, to be put at the 'References' section, looks like:

> [1] Frank Thys. Auf der Spur des Vernichters. In Wolfgang Holbein, editor, *Dinoland*, pp. 353–437, Bergisch Gladbach, August 1997. Bastei Lübbe.

We got this result by using 'old' BibTEX, operating on an 'old' bibliography (.bib) file. The bibliography style used above is plain.bst, that is, items are labelled by numbers, and first names are not abbreviated. This reference is supposed to be put at the end of a document written in English. If a German-speaking plain bibliography style — e.g., dtk.bst, used for the articles of the journal of the DANTE[8] group, *Die TEXnische Komödie* — is chosen, that results in:

> [1] Frank Thys: *Auf der Spur des Vernichters*; in *Dinoland* (Hg. Wolfgang Holbein); S. 353–437; Bergisch Gladbach; Aug. 1997; Bastei Lübbe.

so the stylistic differences between these two examples — for example, '.' after the author's name in English, ':' in German and French — shows that the layout of such references is language-dependent, in the sense that it is influenced by 'national' traditions.

When MlBibTEX parses the entry of Figure 1, the entry is processed as if it was the XML tree given in Figure 2; in fact, it results in the SXML[9] representation of such an XML tree. We can notice that this choice allows us to structure information given in some fields, for example, person names, in the `AUTHOR` and `EDITOR` fields, but also the first and last pages of a story belonging to an anthology, in the

[7] Also used in conjonction with the mlbib package [23] or the natbib package [7], but in MlBibTEX, the corresponding values need not to be surrounded by braces or double-quote characters.

[8] *Deutschsprachige Anwendervereinigung TEX e.V.*

[9] Scheme implementation of XML, described in [20]. See [15] for more details about its use within MlBibTEX's implementation.

```
<nbst:template match="pages">
  <nbst:param name="beginning"/>
  <nbst:param name="ending"/>
  <nbst:value-of select="$beginning"/>
  <nbst:variable name="onepage-elements" select="onepage">
  <nbst:choose>
    <nbst:when test="$onepage-elements">
      <nbst:choose>
        <nbst:when test="count($one-page-elements) = 1"><nbst:text>\bblp</nbst:text></nbst:when>
        <nbst:otherwise><nbst:text>\bblpp</nbst:text></nbst:otherwise>
      </nbst:choose>
      <nbst:apply-templates select="$onepage-elements[1]"/>
    </nbst:when>
    <!--  Otherwise, firstpage element, followed by either the ff or a last page.      -->
    <nbst:otherwise><nbst:apply-templates/></nbst:otherwise>
  </nbst:choose>
  <nbst:value-of select="$ending"/>
</nbst:template>

<nbst:template match="onepage">
  <nbst:param name="first-time" select="true()"/>
  <nbst:variable name="following" select="following-sibling::onepage">
  <nbst:choose>
    <nbst:when test="$first-time"><nbst:call-template name="tie-number"/></nbst:when>
    <nbst:otherwise><nbst:value-of select="."/></nbst:otherwise>
  </nbst:choose>
  <nbst:if test="$following">
    <nbst:text>, </nbst:text>
    <nbst:apply-templates select="$following[1]">
      <nbst:with-param name="first-time" select="false()"/>
    </nbst:apply-templates>
  </nbst:if>
</nbst:template>

<nbst:template match="firstpage | pages-verbatim">   <!--  Putting a non-breaking space character     -->
  <nbst:call-template name="tie-number"/>           <!--  before a small number.                     -->
</nbst:template>

<nbst:template match="ff">
  <nbst:text> \bblff</nbst:text>
</nbst:template>
```

Figure 5: Putting page numbers down in nbst.

PAGES field. Such XML trees are conformant to a DTD[10], an excerpt from which being given in Figure 3. Syntactically, the PAGES field of MlBibTEX allows the specification of:

- a single page: {353},
- a range of pages: {353--457},
- the first page of an unspecified number of consecutive ones: {353+},
- some enumerated pages: {353,439,519},
- otherwise, the value associated with this field is kept *verbatim* and becomes the content of the pages-verbatim element: this content will appear as it is within any predefined bibliography style.

The bibliography styles of BibTEX deal with these different syntactic forms, as it can be seen in Figure 4, but this style of programming seems to us to be some *hack*.

Figure 5 shows how page numbers can be processed using nbst. Many tags and attributes are the same than in XSLT, except for the namespace used as a prefix, which is obviously different. We explain

[10] **D**ocument **T**ype **D**efinition. A DTD defines a document markup model [29, Ch. 5]. The DTD we use is a revised version of what is given in [10].

Entity reference	Character	How to produce it in LaTeX	Numeric entity
`&amp;`	&	`\&`	`&#38;`
`&apos;`	’		`&#39;`
`&emdash;`	—	`---`	`&#151;`
`&endash;`	–	`--`	`&#150;`
`&eol;`	¶[a]	`\newline`	`&#10;`
`&gt;`	<		`&#62;`
`&lt;`	>		`&#60;`
`&nobsp;`[b]		`~`	`&#160;`
`&quot;`	"		`&#34;`

[a] ‘¶’ is a typographic sign for the end-of-line character [2, § 2.85]. In nbst, this entity is used to begin a new line within generated files.

[b] Non-breaking space character.

Table 1: Entities usable in nbst.

later what the parameters `beginning` and `ending` are precisely, but intuitively, we can guess that they are strings to be put before and after the page numbers. Let us notice the use of *variables* — names that may be bound to values — and of *path expressions* in `match` and `select` attributes’ values. Using the `following-sibling` axis allows us to reach the subtrees at the right of the current node and sharing the same parent node, that is particularly useful to implement loops, in the sense of ‘classical’ programming languages. Putting some enumerated pages would be done this way if we express it using a ‘classical’ algorithm:

```
write(tie-number(first(one-page-elements))) ;
loop
  one-page-elements ← rest(one-page-elements) ;
  exit when one-page-elements = ∅ ;
  write(",␣") ; write(first(one-page-elements)) ;
end loop ;
```

Figure 5 shows how this algorithm is put into action by means of a recursive template, matching the first element of page numbers not written yet. This technique is very common in XSLT for iterative algorithms.

Let us focus on the texts generated when these templates are invoked, more precisely, on the content of the `nbst:text` tags: we notice the use of additional LaTeX commands, for example, `\bblp` (resp. `\bblpp`) for one (resp. several) pages. These names originate from bibliography styles generated by the makebst program [3] in interface with the babel package [24, Ch. 9], and are language-dependent. For example, the `\bblp` command is expanded in ‘p.’ for ‘page’ in English and French, in ‘S.’ for ‘*Seite*’ in German. How to organise them is shown in [14, § 2].

```
<nbst:template match="lastpage">
  <nbst:value-of
    select="concat('&endash;',.)"/>
</nbst:template>
```

```
<nbst:template match="lastpage"
               language="french">
  <nbst:value-of select="concat('-',.)"/>
</nbst:template>
```

Figure 6: Default and language-dependent templates.

Special characters can be denoted by entity references, like in XML [29, pp. 48–49]. MlBibTEX knows more predefined character entities than XML — e.g., ‘`&endash;`’, used in Figure 6 — they are summarised in Table 1: for each, we give its name, the corresponding character, the way to produce it in LaTeX if this character is special[11], the decimal number coding it w.r.t. Unicode [33].

Now let us introduce the main difference between XSLT and nbst. When a range of pages is to be given, an *en-dash* character[12] should be put between the first and last page numbers. More precisely, this is the convention for most European languages, including English. But in documents written in French, this character tends to be replaced by a single minus character (‘-’). In our style, this character is put by the template processing the last page number. Figure 6 gives two version of this template: a default version, without the `language`, and another version, suitable for the French language. This `language` attribute does not exist in XSLT; in nbst, a template with it has higher priority than the same template without.

Style for a entry type

As we can read in [24, § 13.6.3], introducing small changes in a bibliography style written using the bst language is quite easy. Writing the whole of a style is a worthwhile exercise: we have to know what has been pushed onto the stack handled by BibTEX, what we can pop from it, possibly after applying the `duplicate$` function when this value is needed afterwards by the program. This language is not modular, we have to take care of such questions from a

[11] MlBibTEX uses it only when the `mode` attribute of the `nbst:output` element (cf. Figure 12) is `LaTeX`. For example, the element:

```
<nbst:text>The Bull &amp; the Spear</nbst:text>
```

produces ‘`The Bull \& the Spear`’ (resp. ‘`The Bull & the Spear`’) if the mode is `LaTeX` (resp. `text`).

[12] That is, a dash as wide as the ‘n’ letter.

```
<inproceedings> ::=
    "\bibitem{" <id> "}¶" <authors> <title> <in-eds-booktitle> [", " <volume-number-series>]
    [", " <pages>] <date-etc> ["¶\newblock " <note> "."] "¶¶" ;
<authors>               ::= <name-list> ".¶\newblock " ;
<editors>               ::= <name-list> ", \bbled, "                   if |<name-list>| = 1  |
                            <name-list> ", \bbleds, "                  if |<name-list>| > 1  ;
<name-list>             ::= <name> {", " <name>} [", \bbland\ " <name> | " \bbletal"] ;
<title>                 ::= change-case(t)(<string>) ".¶\newblock " ;
<booktitle>             ::= "\emph{" <string> "}" ;
<in-eds-booktitle>      ::= "\capitalize\bblin " [<editors>] <booktitle> ;
<volume-number-series>  ::= "\bblvol" <tie-number>_<volume> " \bblof \emph{" <series> "}" |
                            "\bblno" <tie-number>_<number> " \bblin " <series> ;
<pages>                 ::= "\bblp" <tie-number(s)>                    if |<tie-number(s)>| = 1  |
                            "\bblpp" <tie-number(s)>                   if |<tie-number(s)>| > 1  ;
<tie-number(s)>         ::= <non-breaking-space-character> <number(s)>   if <number(s)> < 3  |
                            " " <number(s)>                            if <number(s)> ≥ 3  ;
<date-etc>              ::= [", " <address> ", "] <date> [". " <org-pub>] ". " |
                            [". " <org-pub>] ", " <date>
<org-pub>               ::= [<organisation> ", "] <publisher> ;
```

'|...|' is for the number of elements of a list, '$\overline{\ldots}$' for the length of a string. Cf. Table 1 about the '¶' sign.

Figure 7: How to put information about a story included into an anthology.

function to another, and the use of only global variables reinforces this monolithic way of programming. So, the best method for rewriting a style wholly is to express it using a grammar, according to a *reverse engineering*[13] approach. That is, studying bst styles in order to deduce such a grammar. Of course, such modelling can also be done from documents giving rules for bibliographies' layout, such as [1, § 10] or [2, §§ 15 & 16].

Figure 7 gives all the possible texts for references generated by BibTEX, using a 'plain' style and derived from entries being `@INPROCEEDINGS` type. We do not consider cross-referencing ([22, § B.1.4], [24, § 13.2.5]), not implemented yet in MlBibTEX. These possible texts are expressed with a formalism close to EBNF[14], that is:

- for each non-terminal symbol, enclosed like an XML tag, the expression following the '`::=`' sign and terminated by '`;`' states how it can be expanded;
- the '`|`' sign means an alternative, '`[...]`' is for an optional part, '`{...}`' for zero or more occurrences of its content;
- expressions enclosed by two double quote characters are texts to be put: let us recall that they are part of LATEX input.

Since this grammar does not model texts to be parsed, but texts to be generated, we do not have to be conformant with conditions related to parsing, as that would be the case for a language to be interpreted or compiled. In fact, most of our non-terminal symbols are fields' names of MlBibTEX (e.g., `<title>`) or simple types (e.g., `<string>`). There is some language abuse — for example, the use of functions (e.g., `change-case`[15]) — but we think that such a specification is precise and gives a good overview of the texts to be generated.

So, we are given precise information about the order in which fields' values should be placed. As specified in the file plain.bst, we keep the occurrences of the `\newblock` command, used when the bibliography is to be 'open' — by means of the openbib option of the `\documentclass` command — that is, each block starting on a new line [24, § 12.2.1]. On another point, some keywords, hard-wired in this file, are replaced by multilingual commands of LATEX. By the way, let us remark that we are able

[13] According to the terminology used in Software Enginnering:

- **re-engineering** consists of transforming a program written using an 'old' language into a new program in a more modern language: for example, deriving a C program from source files written in FORTRAN;
- **reverse engineering** is the process of analysing software in order to recover its design of specification.

As stated in [31, Ch. 34], reverse engineering is part of software re-engineering process, in the sense that allows better understanding of a system.

[14] **E**xtended **B**ackus-**N**aur **F**orm. Readers unfamiliar with this formalism can refer to [4] for an introduction. DTD syntax originate from it.

[15] Analogous to the namesake function in BibTEX [25].

```
<nbst:template match="inproceedings">
  <nbst:call-template name="common-pre"/>
  <nbst:variable name="comma-space"
                 select="', '"/>
  <nbst:apply-templates select="author"/>
  <nbst:apply-templates select="title"
                        mode="inproc"/>
  <nbst:call-template name="in-eds-booktitle"/>
  <nbst:call-template
    name="volume-number-series">
    <nbst:with-param name="beginning"
                     select="$comma-space"/>
  </nbst:call-template>
  <nbst:variable name="pages">
    <nbst:apply-templates select="pages">
      <nbst:with-param name="beginning"
                       select="$comma-space"/>
    </nbst:apply-templates>
  </nbst:variable>
  <nbst:call-template name="date-etc">
    <nbst:with-param name="previous"
                     select="$pages"/>
  </nbst:call-template>
  <nbst:apply-templates select="note">
    <nbst:with-param
      name="beginning"
      select="'&eol;\newblock '"/>
    <nbst:with-param name="ending"
                     select="'.'"/>
  </nbst:apply-templates>
  <nbst:call-templates name="common-post"/>
</nbst:template>
```

Figure 8: Building a reference from an `inproceedings` element: program using nbst.

to capitalise the result of such a command when it begins a sentence, by means of the `\capitalize` command[16]. As far as possible, we consider that a sign of ponctuation terminates the written form of a field — for example, the list of authors, ended with a period — but it is not always possible: as another example, the specification of page numbers may be followed by a comma if there is an address, by a period if there is an organisation name. In such a case, the sign of ponctuation is specified before the non-terminal symbol it opens in Figure 7.

```
<nbst:template match="title" mode="inproc">
  <nbst:apply-templates match=".">
    <nbst:with-param name="emf"
                     select="false()"/>
    <nbst:with-param name="retain-capitals"
                     select="false()"/>
  </nbst:apply-templates>
</nbst:template>
```

Figure 9: Putting titles down.

Now the role of the two template parameters `beginning` and `ending`, occurring in Figure 5 is explained. Their use is systematic, as it can be seen in Figure 8, that 'implements' our specification. More generally, we can notice that writing this template matching `inproceedings` elements is direct, once we got a grammar for such references. If we consider Figure 7, the layout for an element (e.g., `<author>`) is implemented by a template with a `match` attribute; if we implement a non-terminal symbol grouping the layout of several elements (e.g., `<in-eds-booktitle>`), a named template does that. The named template `common-pre` opens a reference, by putting the `\bibitem` command [24, § 12.1.2], whereas the `common-post` template closes it. Both may used to insert multilingual directives, for example, the `otherlanguage` environment of the babel package [24, § 9.2.1].

Let us mention a last point about signs of ponctuation: several consecutive ones may conflict. In practice, such a case occurs when a period is to be put after a string ending with an exclamation or question mark, or with a period belonging to an abbreviation. BIBTEX solves this case by means of its function `add.period$` [25], provided that the string has not been popped yet. In XSLT and nbst, a string is output by means of the `value-of` element, unless it is processed within a template that becomes the content of a variable. Thereby the result of this template can be memoized and reused later. Let us look at Figure 8: the string result of invoking the template matching the `pages` element becomes the value of the `pages` variable, which is passed to the named templates `date-etc`.

Refining the way to process `title` elements, let us remark that it depends on the entry type: within the bibliography style plain.nbst, they are put down using italic characters for an entry type being type `@BOOK`, written using roman characters without quotation marks if this type is `@INPROCEEDINGS`. In this last case, we process such an element with a

[16] This command is not predefined in LATEX, it can be defined as follows:

```
\def\capitalize#1{%
 \def\Capitalize##1{\uppercase{##1}}%
 \expandafter\Capitalize#1}
```

cf. [21] for more details about `\expandafter` and the definitions of TEX commands.

```
<nbst:template match="title">
  <nbst:param name="emf" select="true()"/>
  <nbst:param name="quotedbf" select="false()"/>
  <nbst:param name="retain-capitals" select="true()"/>
  <nbst:param name="ending" select="’.&eol;\newblock’"/>
  <nbst:if test="$quotedbf"><nbst:text>\begin{bblquotedtitle}</nbst:text></nbst:if>
  <nbst:if test="$emf"><nbst:text>\emph{</nbst:text></nbst:if>
  <nbst:variable name="title-put">
    <nbst;choose>
      <nbst:when test="$retain-capitals"><nbst:apply-templates/></nbst:when>
      <nbst:otherwise>
        <nbst:apply-templates select="node()[1]">
          <nbst:with-param name="retain-capitals" select="false()"/>
          <nbst:with-param name="no-left-lowercase" select="true()"/>
        </nbst:apply-templates>
        <nbst:apply-templates select="node()[position() &gt; 1]">
          <nbst:with-param name="retain-capitals" select="false()"/>
        </nbst:apply-templates>
      </nbst:otherwise>
    </nbst:choose>
  </nbst:variable>
  <nbst:value-of select="$title-put"/>
  <nbst:if test="$emf"><nbst:text>}</nbst:text></nbst:if>
  <nbst:if test="$quotedbf"><nbst:text>\end{bblquotedtitle}</nbst:text></nbst:if>
  <nbst:call-template name="adjoin-sign">
    <nbst:with-param name="the-string" select="$title-put"/>
    <nbst:with-param name="ending" select="$ending"/>
  </nbst:call-template>
</nbst:template>
```

Figure 10: Putting titles down (*continued*).

`mode` attribute, as shown in Figure 9. The template matching `title` elements without any mode — cf. Figure 10 — allows us to define parameters for ruling the layout and give them default values used when we display the title of a book:

- `emf`: if true, use italic characters;
- `quotedbf`: if true, use language-dependent quotation marks, provided by the `bblquotedtitle` environment (cf. [14, § 2]);
- `retain-capitals`: if false, converting the title to lowercase except at the beginning;
- `ending`: the string to be put after the title. The named template `adjoin-sign` prevents conflict between the last character of the title and the value of `ending`.

As shown in Figure 9, this template with the `mode` attribute set to `inproc` only consists of passing suitable values to the general template of Figure 10. Processing titles according to this `inproc` mode can be redefined for the French language, using French quotation marks, or the German language, using italic characters, as written in Figure 11.

Core of a style

When MlBibTEX builds an XML-like tree with all the entries to be processed, this tree is rooted by an element so-called `mlbiblio`. Figure 12 gives the root element of our ‘new plain’ bibliography style and shows how to process all the entries. Opening the `thebibliography` environment [24, § 12.1.2] is done by the named template `put-preamble`, which may put additional LATEX definitions, especially those included in BibTEX’s preambles [24, § 13.2.4]. Symmetrically, the `putpostamble` template closes the bibliography.

We can also see how entries are sorted before they are dispatched according to their type. Like the namesake element of XSLT, the first occurrence specifies the primary sort key, the second occurrence the secondary sort key, used for elements left unsorted, and so on. The first occurrence could have been specified by:

```
select="author/name[1]/personname/last"
```

that is, sorting entries w.r.t. the family name of the first author, but that would discard organisation

```
<nbst:template match="title" mode="inproc"
               language="french">
  <nbst:apply-templates match=".">
    <nbst:with-param name="emf"
                     select="false()"/>
    <nbst:with-param name="quotedbf"
                     select="true()"/>
  </nbst:apply-templates>
</nbst:template>

<nbst:template match="title" mode="inproc"
               language="german">
  <nbst:apply-templates match=".">
    <nbst:with-param
      name="ending"
      select="';&eol;\newblock'"/>
  </nbst:apply-templates>
</nbst:template>
```

Figure 11: Putting titles down w.r.t. French and German styles.

names as authors. The solution we put in Figure 12 consists of concatenating three strings related to the first author, two of them being always empty:

- the family name, if this name is for a person,
- the sort key of an organisation name, if given,
- the organisation name itself, if the sort key has not been given.

For first authors that are organisation names, only the first occurrence of the `nbst:sort` element is of interest, the others do nothing. When sorting entries w.r.t. names is finished, we sort w.r.t. years, then months. This last sort order can seem to be some *hack* since it uses the interface with Scheme functions (cf. § B), but let us recall that programming such a sort order is very difficult in bst and unused in practice. However, we think that our successive `nbst:sort` elements are clearer than the `presort`, `sortify` and `purify$` functions used within bibliography styles written in bst.

Splitting a style into several files

As abovementioned, the bst language is not modular, and all the definitions for a particular style must be stored in the same file, what is a drawback since several styles share the same definitions. That complicates the mainenance of bibliography styles if some definitions need some enrichment. Besides, it is difficult, when we are studying a style, to determine what is specific or common to other styles. The nbst language includes:

- an `nbst:include` element, to import definitions explicitly from another nbst file;
- *implicit importations*, described in [14, § 3.1].

Hereafter, we show how to spread out the templates we are writing over different files, in order to take as much advantage as possible of implicit importations of nbst. Let us recall that we are developing a new version of the 'plain' bibliography style, that is, the master file is plain.nbst.

- The global.nbst can be viewed as MlBIBTEX's initial library of definitions using nbst: it includes general named templates such as:

  `adjoin-sign` `date-etc` `tie-number`

  as well as template matching the following elements:

  `address`
  `booktitle`
  `ff`
  `firstpage`
  `lastpage`
  `note`
  `one-page`
  `orgnization`
  `pages`
  `pages-verbatim`
  `publisher`
  `title`

  Putting more templates in this file may seem to be of interest, but let us recall that in nbst, imported templates have the same priority than other elements[17]: so 'global' elements cannot be redefined[18], unless adding a `language` or `mode` attribute to the redefinition.
- Of course, the plain.nbst file — the master file for this bibliography style — must include its root (`nbst:bst`) element and the 'main' template matching an `mlbiblio` element, given in Figure 12. The following named templates, related to references' labels, should be included in this file, too:

  `common-post`
  `common-pre`
  `put-postamble`
  `put-preamble`

  The layout of the following element depends on the bibliography style, so the corresponding templates have to be stored in the plain.nbst file:

  `author`
  `editor`
  `inproceedings`
  `number`
  `series`
  `volume`

  as well as the named templates, for the same reason:

  `in-eds-booktitle`
  `org-pub`
  `volume-number-series`
- The 'French' definition of the template matching a `lastpage` element (cf. Figure 6) is general for French-speaking styles, not directly related to 'plain' styles, so we place it onto the

[17] This is not the case in XSLT if the `xsl:import` element is used.

[18] More exactly, if there is conflict between templates, it is unpredictible to know which will be run.

```
<nbst:bst version="1.3" id="plain" xmlns:nbst="http://lifc.univ-fcomte.fr/~hufflen/mlbibtex">

  <nbst:output method="LaTeX" encoding="ISO-8859-1"/>
  <!--  This encoding allows accented letters of Western European Languages [5, Table C.4].     -->

  <nbst:template match="mlbiblio">
    <nbst:call-template name="put-preamble">
      <nbst:with-param name="longest-label" select="count(*)"/>
    </nbst:call-template>
    <nbst:apply-templates>
      <nbst:sort select="concat(author/name[1]/personname/last,
                                author/name[1]/othername/@sortingkey,
                                author/name[1]/othername[not(@sortingkey)])"/>
      <nbst:sort select="author/name[1]/personname/first"/>
      <nbst:sort select="author/name[1]/personname/von"/>
      <nbst:sort select="author/name[1]/personname/junior"/>
      <nbst:sort select="year" data-type="number"/>
      <nbst:sort select="call(month-position,month)" data-type="number"/>
    </nbst:apply-templates>
    <nbst:call-template name="put-postamble"/>
  </nbst:template>

  ...

</nbst:bst>
```

Figure 12: Root element for a program in nbst — Organising all the entries to generate references.

-french.nbst file, that is, the file grouping the general definitions for the French language.

- On the contrary, the French and German redefinitions of the template matching `title` elements in `inproc` mode (cf. Figure 11) belong both to the 'plain' bibliography style so they should be put into the files plain-french.nbst and plain-german.nbst.

Conclusion

We think that when a new tool or a new programming language is developed, its conceptor(s) should provide methodology and advice about it. Often teachers of programming languages notice that students may program badly in a good language. Let us go back to BibTEX, we personally missed — in the past, a long time before we decided to develop MlBibTEX — a didactic introduction to the bst language like [28]. Likewise, an overview for writers of LATEX extensions such as [24, Appendix A] was missing for a long time.

In this article, we have not shown all the features of MlBibTEX. For example, we have not gone thoroughly into multilingual features — in order to show that our approach was mostly suitable for designing styles using xslt, too — and 'new plain' style was implicitly supposed to be language-dependent [13], that is, each reference is expressed using the language's entry. In fact, our goal was to show that nbst allowed us to write bibliography styles in elegant manner, provided that we are given a precise specification of what to put. So we are able to build a solid basis for a style, and people could easily enrich it with new language-dependent templates by using MlBibTEX's implicit importation.

Now we are rewriting predefined bibliography styles of BibTEX. Most of them have already been redesigned, but this work is not finished yet at the time we finish writing this article. We hope that these explanations would help people enrich these new styles, especially in order to adapt them to other languages.

Acknowledgements

Thanks to Volker R. W. Schaa, who proof-read the German translation of the abstract.

A bst vs nbst

A precise comparison between bst and nbst is difficult, since these two languages belong to very different programming paradigms. The former is based on handling a stack (see [28] for a didactic introduction to this aspect), the latter encourages rule-based programming. They do not belong to the same time, either: the former has been influenced by assembly

bst expression	"Equivalent" expression in nbst	Kind[a]
$\mathcal{I}_1$ $\mathcal{I}_2$ >	$\mathcal{I}_1{}^\natural$ &gt; $\mathcal{I}_2{}^\natural$	P
$\mathcal{I}_1$ $\mathcal{I}_2$ <	$\mathcal{I}_1{}^\natural$ &lt; $\mathcal{I}_2{}^\natural$	P
$\mathcal{I}_1$ $\mathcal{I}_2$ =	$\mathcal{I}_1{}^\natural$ = $\mathcal{I}_2{}^\natural$	P
$\mathcal{S}_1$ $\mathcal{S}_2$ =	$\mathcal{S}_1{}^\natural$ = $\mathcal{S}_2{}^\natural$	P
$\mathcal{I}_1$ $\mathcal{I}_2$ +	$\mathcal{I}_1{}^\natural$ + $\mathcal{I}_2{}^\natural$	P
$\mathcal{I}_1$ $\mathcal{I}_2$ -	$\mathcal{S}_1{}^\natural$ - $\mathcal{S}_2{}^\natural$	P
$\mathcal{S}_1$ $\mathcal{S}_2$ *	concat($\mathcal{S}_1{}^\natural$,$\mathcal{S}_2{}^\natural$)	P

$\mathcal{S}$ add.period$	<nbst:call-template name="adjoin-sign">
	  <nbst:with-param name="the-string" select="$\mathcal{S}^\natural$"/>
	  <nbst:with-param name="ending" select="'.'"/>
	</nbst:call-template>
		E^b

$\mathcal{S}$ "t" change.case$	concat(substring($\mathcal{S}^\natural$,1,1),lowercase(substring($\mathcal{S}^\natural$,2)))	P^c
$\mathcal{S}$ "l" change.case$	lowercase($\mathcal{S}^\natural$)	P
$\mathcal{S}$ "u" change.case$	uppercase($\mathcal{S}^\natural$)	P
$\mathcal{S}$ chr.to.int$	(char->integer $\mathcal{S}^\natural$)	S
cite$	@id	P
$\mathcal{L}$ empty$	not(string($\mathcal{L}^\natural$))	P

$\mathcal{I}$ $\mathcal{F}_1$ $\mathcal{F}_2$ if$	<nbst:choose>
	  <nbst:when test="$\mathcal{I}^\natural$ &gt; 0">$\mathcal{F}_1{}^\natural$</nbst:when>
	  <nbst:otherwise>$\mathcal{F}_2{}^\natural$</nbst:otherwise>
	</nbst:choose>
		E

$\mathcal{I}$ int.to.chr$	(integer->char $\mathcal{I}^\natural$)	S
$\mathcal{I}$ int.to.str$	string($\mathcal{I}^\natural$)	P
$\mathcal{L}$ missing$	not($\mathcal{L}^\natural$)	P
newline$	<nbst:text>&eol;</nbst:text> or <nbst:value-of select="'&eol;'"/>	E
$\mathcal{S}$ num.names$	count(name) if name($\mathcal{S}^\natural$) $\in$ {author, editor}	P
preamble$	@preamble	P
$\mathcal{S}$ purify$	call(bst-purify,$\mathcal{S}^\natural$)	P^d
quote$	<nbst:text>&quot;</nbst:text> or <nbst:value-of select="'&quot;'"/>	E
$\mathcal{S}$ $\mathcal{I}_1$ $\mathcal{I}_2$ substring$	substring($\mathcal{S}^\natural$,$\mathcal{I}_1{}^\natural$,$\mathcal{I}_2{}^\natural$) if $\mathcal{I}_1 > 0$ substring($\mathcal{S}^\natural$,string-length($\mathcal{S}^\natural$) $+ \mathcal{I}_1{}^\natural - \mathcal{I}_2{}^\natural + 2$,$\mathcal{I}_2{}^\natural$) if $\mathcal{I}_1 < 0$	P^c
$\mathcal{S}$ text.length$	string-length($\mathcal{S}^\natural$)	P
$\mathcal{S}$ $\mathcal{I}$ text.prefix$	substring($\mathcal{S}^\natural$,1,$\mathcal{I}^\natural$)	P^c
type$	name()	P
$\mathcal{S}$ warning$	<nbst:warning>$\mathcal{S}^\natural$</nbst:warning>	E
$\mathcal{S}$ width$	(tex-width $\mathcal{S}^\natural$)	S^e
$\mathcal{S}$ write$	<nbst:value-of select="$\mathcal{S}^\natural$"/>	E

[a] Qualifies the given expression in nbst: 'E' is for 'element', 'P' for 'path expression', 'S' for 'Scheme expression'.
[b] The adjoin-sign is included in MlBibTEX's initial library.
[c] Let us recall that indexing strings is one-based in XPath and nbst, whereas it is zero-based in Scheme.
[d] This Scheme function is given in Figure 13. Useless in practice (see Figure 8)!
[e] Not implemented yet (always returns "0").

Table 2: Translation of most bst statements given in [24, Table 13.8]

languages, the latter has taken advantage of a modern langage, suitable for handling documents and designed for a large purpose.

Some statements of bst are not really translatable into nbst: for example, the assignment (':='), because nbst is like a purely functional language, in the sense that a variable — or a parameter — cannot be changed, once it has been given a value. On the other hand, nbst allows recursive templates, like in XSLT, what is useful for iterative programming (cf. Figure 5) and replaces the while$ function of bst.

The call.type$ function of bst does not have a direct equivalent, either: such an operation is performed by pattern-matching by means of the match attribute of suitable nbst:template elements. The

```
(define (bst-purify string-0)
  (let thru ((index (- (string-length string-0) 1))
             ;; Current index, we are going backward. The second argument allows us to accumulate retained
             ;; characters in a list, we begin with an empty list:
             (accumulator '()))
    (if (negative? index)
        ;; The string has been processed, we convert the list of accumulated characters into a string:
        (list->string accumulator)
        (thru (- index 1) (let ((current-char (string-ref string-0 index)))
                            ;; Discarding it if it is not alphanumeric:
                            (if (or (char-alphabetic? current-char) (char-numeric? current-char))
                                (cons current-char accumulator)
                                accumulator))))))
```

Figure 13: Scheme function implementing the bst function `purify$`.

`format.name$` function is replaced by handling path expressions like in XPath for subtrees for authors and editors.

Table 2 is an attempt to express the relationship between bst statements and corresponding realisations in nbst. In fact, it emphasises which statements are easily translatable, which are not. This table does not include bst functions such as ':=', `while$`, `call.type$`, `skip$`. Likewise, we did not put bst functions directly related to BIBTEX's stack management: `duplicate$`, `stack$`, `swap$`, `top$`.

For the other bst functions, we make precise its operands: $\mathcal{I}$ is for an integer, $\mathcal{S}$ for a string, $\mathcal{L}$ for any value, $\mathcal{F}$ for a function. When several operands are the same type, we use indices. We use the '...$^\natural$' notation to mean 'the translation of an operand in nbst': for example, the `if$` function of bst pops three values from the stack, the translation of the first should be used inside the value of a `test` attribute, the others should be translated into nbst elements put as contents of an `nbst:if` element.

As it can be seen in Table 2, the direct translation of some statements needs to call functions directly written in Scheme: we put them for sake of completeness, but in practice, most of these functions are useless when a style is wholly rewritten using nbst (cf. § B). Last, let us remark that in the path expressions given in this table — `@id` and `@preamble` — the current node is supposed to be the node for an entry (`inproceedings`, `book`, ...)

B Interface with Scheme

Path expressions used within nbst include calls to external functions written in Scheme and returning strings. The syntax is:

`call(`*`function-name`*`,`arg_1`,...,`arg_n`)`

where ***`function-name`*** is the function's name, applied to $\mathbf{arg}_1, \ldots, \mathbf{arg}_n$ ($n \in \mathbb{N}$). Now we got some experience in writing bibliography styles, and as far as we know, there are three reasons for using such functions within bibliography style files:

- functions related to TEX's features: for example, returning the width of a string, expressed in TEX's units (cf. Table 2), as another example, searching LATEX source files: for instance, we have to do that in order to know the document's language[19];
- operations that would be tedious with the functions of XPath's library: an example appearing in Table 2 is the `bst-purify` function;
- functions used to sort entries: e.g., the function `month-position`, that allows the sort of month names according to the chronological order, used in the template given in Figure 8.

In Figure 13, we give the exact equivalent for the `purify$` function of bst, in order to give some idea about how to deal with strings in Scheme. Let us remark that this operation — used in BIBTEX to build strings to be sorted lexicographically — is useless practically since it is better to use successive `nbst:sort` elements as we show in Figure 12.

In addition to the `bst-purify` function, we give a second example written in Scheme in Figure 14: the `month-position` function, used to sort month names, as shown in Figure 12. As abovementioned, this way may be thought as *ad hoc* method, nevertheless, let us remark that such a sort is not provided by 'old' BIBTEX.

[19] See [16] for more details about this problem. MlBIBTEX also searches auxiliary (.aux) files produced by LATEX, but not whilst a bibliography style is applied.

```
(define month-position
  (let ((month-name-list
         '("jan" "feb" "mar" "apr" "may" "jun" "jul" "aug" "sep" "oct" "nov" "dec")))
    (lambda (string-0)
      (let thru ((month-name-list-0 month-name-list)
                 (current-position 0))
        (if (or (null? month-name-list-0)
                ;; This way, elements with a non-recognised or empty month name will be put after those with
                ;; an actual month name after the sorting operation.
                (string=? (car month-name-list-0) string-0))
            (number->string current-position)   ; Final result as a string.
            (thru (cdr month-name-list-0) (+ current-position 1)))))))
```

Figure 14: Scheme function used to sort month names by sorting corresponding positions.

References

[1] Judith BUTCHER: *Copy-Editing. The Cambridge Handbook for Editors, Authors, Publishers.* 3rd edition. Cambridge University Press. 1992.

[2] *The Chicago Manual of Style.* The University of Chicago Press. The 14th edition of a manual of style revised and expanded. 1993.

[3] Patrick W. DALY: *Customizing Bibliographic Style Files.* Version 3.2. February 1999. Part of BIBTEX's distribution.

[4] Lars Marius GARSHOL: BNF *and* EBNF*: What Are They and How Do They Work?* July 2003. `http://www.garshol.priv.no/download/text/bnf.html`.

[5] Michel GOOSSENS and Sebastian RAHTZ, with Eitan M. GURARI, Ross MOORE and Robert S. SUTOR: *The LaTeX Web Companion.* Addison-Wesley Longmann, Inc., Reading, Massachusetts. May 1999.

[6] Vidar Bronken GUNDERSEN and Zeger W. HENDRIKSE: *BIBTEX as* XML *Markup.* January 2003. `http://bibtexml.sourceforge.net`.

[7] Harald HARDERS: „Mehrsprachige Literaturverzeichnisse: Anwendung und Erweiterung des Pakets babelbib“. *Die TeXnische Komödie*, Bd. 4/2003, S. 39–63. November 2003.

[8] Erik VAN HERWIJNEN: *Practical* SGML. Interpharm Press. December 1994.

[9] Jean-Michel HUFFLEN: “MlBIBTEX: a New Implementation of BIBTEX”. In: *EuroTeX 2001*, p. 74–94. Kerkrade, The Netherlands. September 2001.

[10] Jean-Michel HUFFLEN: “Multilingual Features for Bibliography Programs: From XML to MlBIBTEX”. In: *EuroTeX 2002*, p. 46–59. Bachotek, Poland. April 2002.

[11] Jean-Michel HUFFLEN: “Mixing Two Bibliography Style Languages”. In: LDTA *2003*, Vol. 82.3 of ENTCS. Elsevier, Warsaw, Poland. April 2003.

[12] Jean-Michel HUFFLEN: “European Bibliography Styles and MlBIBTEX”. TUG*boat*, Vol. 24, no. 3. EuroTeX 2003, Brest, France. June 2003.

[13] Jean-Michel HUFFLEN: “MlBIBTEX's Version 1.3”. TUG*boat*, Vol. 24, no. 2, p. 249–262. July 2003.

[14] Jean-Michel HUFFLEN: “Making MlBIBTEX Fit for a Particular Language. Example of the Polish Language”. *Biuletyn* GUST, Vol. 21, p. 14–26. 2004.

[15] Jean-Michel HUFFLEN: “A Tour around MlBIBTEX and Its Implementation(s)”. *Biuletyn* GUST, Vol. 20, p. 21–28. In *BachoTeX 2004 conference.* April 2004.

[16] Jean-Michel HUFFLEN: “MlBIBTEX: beyond LaTeX”. In: *International Conference on TeX,* XML*, and Digital Typography*, Vol. 3130 of LNCS, p. 203–215. Springer, Xanthi, Greece. August 2004.

[17] Jean-Michel HUFFLEN: *Multilingual Bibliography Styles:* nbst *vs* XSLT. To appear in Proc. GUIT conference, Pisa. October 2004.

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

[19] Richard KELSEY, William D. CLINGER, Jonathan A. REES, 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*. February 1998. `http://www.cs.indiana.edu/scheme-repository/`.

[20] Oleg KISELYOV: "A Better XML Parser through Functional Programming". In: *4th International Symposium on Practical Aspects of Declarative Languages*, Vol. 2257 of LNCS. Springer. 2002.

[21] Donald Ervin KNUTH: *Computers & Typesetting. Vol. A: the TEXbook*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.

[22] Leslie LAMPORT: *LATEX. A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.

[23] Wenzel MATIASKE: *Multilinguale Zitierformate*. Oktober 1995. `CTAN:macros/latex/contrib/supported/mlbib/`.

[24] Frank MITTELBACH and Michel GOOSSENS, with Joannes BRAAMS, David CARLISLE, Chris A. ROWLEY, Christine DETIG and Joachim SCHROD: *The LATEX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.

[25] Oren PATASHNIK: *Designing BIBTEX Styles*. February 1988. Part of BIBTEX's distribution.

[26] Oren PATASHNIK: *BIBTEXing*. February 1988. Part of BIBTEX's distribution.

[27] Chris PUTNAM: *Bibliography Conversion Utilities*. February 2005. `http://www.scripps.edu/~cdputnam/software/bibutils/bibutils.html`.

[28] Bernd RAICHLE: *Tutorium: Einführung in die BIBTEX-Programmierung*. Handouts für DANTE 2002. Februar 2002.

[29] Erik T. RAY: *Learning* XML. O'Reilly & Associates, Inc. January 2001.

[30] John E. SIMPSON: *XPath and XPointer*. O'Reilly & Associates, Inc. August 2002.

[31] Ian SOMMERVILLE: *Software Engineering*. 5th edition. Addison-Wesley Publishing Company. 1996.

[32] George SPRINGER and Daniel P. FRIEDMAN: *Scheme and the Art of Programming*. The MIT Press, McGraw-Hill Book Company. 1989.

[33] THE UNICODE CONSORTIUM: *The Unicode Standard Version 4.0*. Addison-Wesley. August 2003.

[34] Doug TIDWELL: XSLT. O'Reilly & Associates, Inc. August 2001.

[35] W3C: XML *Path Language (XPath). Version 1.0*. W3C Recommendation. Edited by James Clark and Steve DeRose. November 1999. `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[36] W3C: XSL *Transformations (*XSLT*). Version 1.0*. W3C Recommendation. Edited by James Clark. November 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.

[37] W3C: *Extensible Stylesheet Language (*XSL*). Version 1.0*. W3C Recommendation. Edited by James Clark. October 2001. `http://www.w3.org/TR/2001/REC-xsl-20011015/`.

[38] Norman WALSH and Leonard MUELLNER: *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc. October 1999.