mlBibTeX: Reporting the Experience
Jean-Michel Hufflen

To cite this version:
Jean-Michel Hufflen. mlBibTeX: Reporting the Experience. TUGB, 2007, pp.157–162. hal-00644467

HAL Id: hal-00644467
https://hal.science/hal-00644467
Submitted on 24 Nov 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
MlBibTeX: Reporting the Experience

Jean-Michel HUFFLEN
LIFC (EA CNRS 4157)
University of Franche-Comté
16, route de Gray
25030 BESANÇON CEDEX
FRANCE
hufflen@lifc.univ-fcomte.fr
http://lifc.univ-fcomte.fr/~hufflen

Abstract
This article reports how the different steps of the MlBibTeX project were conducted until the first public release. We particularly focus on the problems raised by reimplementing a program (BibTeX) that came out in the 1980’s. Since this time, implementation techniques have evolved, new requirements have appeared, as well as new programs within TeX’s galaxy. Our choices are explained and discussed.

Keywords TeX, LaTeX, BibTeX, reimplement, reverse engineering, implementation language, program update.

Streszczenie
Artykuł omawia realizację poszczególnych kroków przedsięwzięcia MlBibTeX, w czasie do przedstawienia pierwszej publicznej wersji. W szczególności skupiamy się na problemach powstałych przy reimplementacji programu (BibTeX), powstałego w latach 80 zeszłego wieku. Od tego czasu rozwinęły się techniki implementacyjne, powstały nowe wymagania oraz nowe programy w świecie TeX-owym. Przedstawiamy i dyskutujemy dokonane wybory.

Słowa kluczowe TeX, LaTeX, BibTeX, reimplementacja, reverse engineering, język implementacji, aktualizacja programu.

0 Introduction
In 2003, TeX’s 25th anniversary was celebrated at the TUG conference, held at Hawaii [1]. BIBTeX [28] and BibTeX [35]—the bibliography processor usually associated with the BIBTeX word processor—are more recent, since they came out in the 1980’s, shortly after TeX. All are still widely used, such longevity being exceptional for software. However, these programs are quite ageing. Of course, recent versions have incorporated many features absent from the first versions, what proves the robustness of these systems. Nevertheless, they present some limitations due to the original conception, and a major reimplementation may be needed to integrate some modern requirements. In addition, interactive word processors made important progress and are serious rivals, even if they do not yield typesetting of such professional quality. That is why some projects aim to provide new programs, based on TeX & Co.’s ideas. A first representative example is the BIBTeX 3 project [32], a second is NtS [27]. MlBibTeX— for ‘MultiLingual BibTeX’— belongs to such projects. Let us recall that this program aims to be a ‘better BibTeX’, especially about multilingual features. For a end-user, MlBibTeX behaves exactly like ‘classical’ BibTeX: it searches bibliography data base (.bib) files for citation keys used throughout a document and arranges the references found into a .bbl file suitable for LaTeX, w.r.t. a bibliography style. MlBibTeX is written in Scheme, uses XML as a central format: when entries of .bib

---

2 Concerning TeX, an additional point is that TeX’s development has been frozen by its author, Donald E. Knuth [26]. If incorporating new ideas to a ‘new TeX’ leads to a major reimplementation, the resulting program must be named differently.

3 The version used is described in [24].

4 EXtensible Markup Language. Readers interested in an introductory book to this formalism can consult [37].

TUGboat, Volume 0 (2060), No. 0 — Proceedings of the 2060 Annual Meeting
files are parsed, they result in an XML tree. Bibliography styles taking advantage as far as possible of MiBiT\TeX 's new features are written using \texttt{bst},\footnote{New Bibliography STyles.} a variant of XSLT\footnote{eXtensible Language Stylesheet Transformations, the language of transformations used for XML documents [43].} described in [15]. The \texttt{bst} language [34], based on handling a stack and used for writing bibliography styles of Bib\TeX{}, can be used in a compatibility mode [20].

We think that the experience we have got in developing MiBiT\TeX{} may be useful for other analogous projects. In a first section, we briefly recall the chronology of this development. As it could be seen, this development has not been linear, and the two next following sections focus on the problems we had to face. We explain how we have determined which criteria are accurate when a programming language is to be chosen for such an application. Then we show how the compatibility with ‘old’ data and the integration of modern features should be managed.

1 MiBiT\TeX{}’s Chronology

Oct. 2000 MiBiT\TeX{}’s design begins: the syntax of .bib files is enriched with multilingual annotations. Version 1.1’s prototype is written using the C programming language and tries to reuse parts of ‘old’ Bib\TeX{}’s program as far as possible.

May 2001 The first article about MiBiT\TeX{} is [9]. Later, the experience of developing MiBiT\TeX{}’s Version 1.1 is described in [10].

May 2002 By discussing with some people at the EuroBacho\TeX{} conference, we realise that the conventions for bibliography styles are too diverse, even if we consider only those of European countries. We realise that this first approach is quite unsuitable, unless defining a new version of the \texttt{bst} language. So we decide to explore two directions. First, we develop a questionnaire about problems and conventions concerning bibliography styles used within European countries. Second, we begin a prototype in Scheme implementing the \texttt{bst} language [11]. Initially, this prototype is devoted to experiments about improving this language in a second version (1.2).

Jan. 2003 Version 1.2 is stalled. The new version (1.3) is built out of XML formats. The \texttt{bst} language is designed and presented at [12, 13]. We explain in [14] how the results of our questionnaire have influenced this new direction.

Feb. 2004 It appears to us that MiBiT\TeX{} should be developed using a very high-level programming language, higher than C. So we consider again the prototype in Scheme, we sketched in 2002. SXML\footnote{Scheme implementation of XML.} [25] is chosen as the representation of XML texts in Scheme. Some parts of MiBiT\TeX{} are directly reprogrammed from C to Scheme. About the other parts, this prototype is a good basis for much experiment [16].

Nov. 2004 The version written in C is definitely dropped out, whereas the version in Scheme is adapted in order to get much efficiency and becomes the ‘official’ MiBiT\TeX{} [18].

Sep. 2005 We decided to freeze MiBiT\TeX{}’s design and concentrate only on finishing programming. Many Scheme functions are rewritten in conformity to SRFIs\footnote{Scheme Requests for Implementation. That is an effort to coordinate libraries and other additions to the Scheme language between implementations.} [39].

May 2006 A working version is almost finished, except for the interface with the kpathsea library.

May 2007 Public availability of MiBiT\TeX{}’s Version 1.3.

Besides, let us make precise that MiBiT\TeX{} is not our only task. As an Assistant Professor in our university, we teach Computer Science. We also participate in other projects. As a consequence, MiBiT\TeX{}’s development has been somewhat anarchic: we hardly worked on it for two or three months, put it aside for one or two months, and so on. Last, we have supervised some student projects about graphical tools around MiBiT\TeX{} [2, 8], programmed using Ruby [31], but concerning the development of the MiBiT\TeX{} program itself, we have done it alone.

2 Choice of an implementation language

There are several programming paradigms: imperative, functional, and logic programming. There are also several ways to implement a programming language: interpretation and compilation. Some paradigms are more accurate, according to the domain of interest. Likewise, some interpreted languages are more accurate if you want to program a prototype quickly and are just interested in performing some experiment.\footnote{That is the case for the graphical tools around MiBiT\TeX{} programmed in Ruby by our students [2, 8].} But compiled languages are often preferable if programs’ efficiency is crucial. Besides, the level of a programming language has some influence on development: in a high-level language, low-level details of structures’ implementation do not have to be made explicit, so development is
quickier, and programs that result in are more concise, nearer to a mathematical model.

In addition to these general considerations, let us recall that we aim to replace an existing program by a new one. This new program is supposed to do better than the ‘old’ one. ‘To do better’ may mean ‘to have more functionalities, more expressive power’, but for sake of credibility, it is preferable for the new program to be as efficient as the ‘old’ one. Let us not forget that \( \TeX \) and \( \texttt{Bib}\TeX \) are written using an old style of programming — more precisely, a monolithic style used in the 1970’s-1980’s — based on global variables mainly, without abstract data types. Choosing a language implemented efficiently is crucial: as a counter-example, \( \texttt{M\underline{e}\underline{c}\underline{a}m\underline{s}} \) [30], written using \texttt{Java}, has been reported 100 times slower than \( \TeX \) [42, § 5].

That is why we wrote \texttt{M\underline{I}B\underline{I}\underline{H}\underline{T}E\underline{X}}’s first version using \texttt{C}, because of its efficiency. In addition, this language is portable on most operating systems. And to make our program modular, we defined precise rules for naming procedures [10, § 3]. But two problems appeared.

First, \texttt{M\underline{I}B\underline{I}\underline{H}\underline{T}E\underline{X}}’s development has not been a daily task, as abovementioned. Even if we are personally able to program large applications in \texttt{C}, it was difficult to put aside a \texttt{C} program and resume it later: from this point of view, \texttt{C} is not a very high-level language. Besides, let us not forget that we are working within an open domain, as natural languages are. Some change may be needed because of new features concerning languages that had not yet been integrated into \texttt{M\underline{I}B\underline{I}\underline{H}\underline{T}E\underline{X}}’s framework. The higher the level, the more easy such change will be applied.

Second, we want end-users of \texttt{M\underline{I}B\underline{I}\underline{H}\underline{T}E\underline{X}} to be able to influence the behaviour of this program. For example, many \texttt{Bib}\TeX users put \( \texttt{L}t\texttt{E} \texttt{T} \texttt{E} \texttt{X} \) commands inside values associated with fields of \texttt{.bib} files, in order to increase expressive power within bibliographical data. These users should be able to specify how to handle such commands when \texttt{.bib} files are converted into XML trees. In particular, that is useful if \texttt{M\underline{I}B\underline{I}\underline{H}\underline{T}E\underline{X}} is used to produce outputs for other word processors than \( \texttt{L}t\texttt{E} \texttt{T} \texttt{E} \texttt{X} \) [21]. How to do that in \texttt{C}? unless defining a mini-language to express such functions? In this case, using a script language is a better choice . . . provided that this language is efficient. Another choice is a \texttt{Lisp} dialect, as did in \texttt{emacs} [40]: end-users can customise \texttt{emacs’} behaviour by expressions using the \texttt{Emacs Lisp} language [30]. This choice is homogeneous: the whole of the \texttt{emacs} program is expressed using \texttt{Emacs Lisp}, except for the implementation of low-level functionalities.

Finally, our choice was \texttt{Scheme}, the modern dialect of \texttt{Lisp}. We confess that we are personally attracted by functional programming languages, because they can abstract procedures as well as data: in this sense, they are very high-level programming languages. Concerning \texttt{Scheme}, it seems to us to be undebatable that it has very good expressive power. In addition, it allows some operations to be programmed ‘impurely’, by side effects, like in imperative programming, in order to increase efficiency. However, we use this feature parsimoniously, on local variables, because it breaks principles of functional programming. We have defined precise rules for naming variables, as we did in \texttt{C} for the first version, in order to emphasise the modular decomposition of our program [19]. Last but not least, \texttt{Scheme} programs may be interpreted — when software is being developed — or compiled, in which case they are more efficient. As a good example of \texttt{Scheme} implementation, \texttt{bigloo} [38] compiles \texttt{Scheme} functions by transforming them into \texttt{C} functions, then these \texttt{C} functions are compiled, in turn.

If we compare the two developments in \texttt{C} and \texttt{Scheme}, the latter is better, as it is expected from a very high-level programming language. But programming an application related to \( \TeX \) using a language other than \( \TeX \) reveals a drawback: the \texttt{kpathsea} library [3] is written in \texttt{C}. Let us recall that \texttt{kpathsea} implements functions navigating through the TDS.\footnote{\texttt{TEX} Directory Structure.} In particular, such functions localise the files containing the specification of a class for a \( \texttt{Bib}\TeX \) document or a bibliography style when \( \texttt{Bib}\TeX \) runs. If there is a compatibility mode, for ‘old’ bibliography styles written in \texttt{bst}, the functions of this compatibility mode should be able to localise such files, too. Likewise, ‘new’ bibliography styles written in \texttt{nbst}, should be localised by means of an analogous method. This implies that the language — or, at least, an implementation of the language — used for our software includes an interface with \texttt{C}.

Of course, what we expose above proceeds from general considerations. After all, we do not know if \( \texttt{Bib}\TeX \) [4] — a successor of \( \texttt{Bib}\TeX \) based on \texttt{Java}, bibliography styles are written in \texttt{Java} — is very less efficient than \( \texttt{Bib}\TeX \). This may not be the case. The advantages of script languages in such development appear if we consider \texttt{Bibulus} [45], another successor of \( \texttt{Bib}\TeX \), written using \texttt{Perl}.\footnote{\texttt{Practical Extraction Report Language}. A didactic introduction to this language is [44].} It has developed quicker than \( \texttt{M\underline{I}B\underline{I}\underline{H}\underline{T}E\underline{X}} \), but is
‘less’ multilingual and uses Bib\TeX when it runs. That is, Bibl\TeX does not replace Bib\TeX wholly, as MiBib\TeX attempts to do. In addition, there is an example where the need of a programming language with higher level than C appeared: the project of moving $\Omega$ — a successor of T\TeX — into a C++ platform [36].

We personally think that an implementation of N\TeX\S in Common Lisp [41] — what was planned initially — would have been preferable. As mentioned in [46], the object-oriented features of Common Lisp (CLOS\textsuperscript{13}) have been added to the language’s basis — like C++ object-oriented functionalities have been added to C — but the language itself is not actually object-oriented. In [46], this point is viewed as a drawback. First, we personally think that everything is not an object, from a point of view related to conception. Second, Common Lisp, even if it is a functional programming language, allows some operations to be performed more efficiently by means of side effects, like Scheme.\textsuperscript{14} But Common Lisp’s standard does not specify an interface with C, like Scheme’s, although some implementations provide this service. However, we personally prefer Scheme, simpler and more modern.

3 Choice of strategy

3.1 Languages

T\TeX & Co. have been wonderful programs since the date they came out. However, they behave very nicely, but syntaxes are quite archaic. T\TeX’s is not date they came out. However, they behave very

Tex’s, although some implementations provide this service. However, we personally prefer Scheme, simpler and more modern.

3.2 New services

Now it is admitted that composite tasks are not to be done by a monolithic program, but by means of a cooperation among several programs. From this point of view, the cooperation between L\TeX and Bib\TeX is exemplar. But Bib\TeX is too strongly related to L\TeX. Bib\TeX can be used to build bibliography for Con\TeXt documents, but because this word processor belongs to T\TeX family. On the contrary, writing a converter from Bib\TeX to HTML\textsuperscript{15} by means of the bst language is impossible without loss of quality: for example, the unbreakable space character is represented by ‘\char127’ as in T\TeX — when names are formatted [22], and this convention cannot be changed.\textsuperscript{16} We see that such problems can be avoided by considering an XML-like language as a central format. In our case, generating bibliographies according to other formats than Con\TeXt’s should be easy since the L\TeX commands end-users put into .bib files are removed when these files are parsed. This point is detailed in [17, 21].

4 Conclusion

Last but not least, we have enjoyed to design and implement MiBib\TeX, even if this development back-tracked several times. In addition, we think that this development shows the difficulties related to such a

\textsuperscript{13} Common Lisp Object System.

\textsuperscript{14} Emacs Lisp, too, and the programs of emacs largely use this feature.

\textsuperscript{15} HyperText Markup Language. Readers interested in an introduction to this language can refer to [33].

\textsuperscript{16} In fact, there are such converters, an example being Bib\TeX2HTML [5], written using Objective Caml [29], a functional programming language.
task. Two parts have to be managed in parallel. The first part is reverse engineering, that is, guessing the conception from the program. The second: enlarging what already exists. In comparison with ‘classical’ development of a new program from scratch, tests concerning the compatibility mode are easy to perform: just comparing what is given by the two programs, the ‘old’ one becoming an oracle. But reaching homogeneous conception is not obvious if we want to keep backward compatibility. Nevertheless, we hope that we have done some satisfactory work.

5 Acknowledgements

Many thanks to Jerzy B. Ludwichowski, who has written the Polish translation of the abstract.

References


Mi\TeX{}: Reporting the Experience


[34] Oren Patashnik: Designing B\TeX{} Styles. February 1988. Part of the B ih\TeX{} distribution.


