

Study of Music Notation Description Languages

Michael Droettboom

April 26, 2000

First Draft

Abstract

This document compares two music notation description languages, GUIDO and MUsic DEscription LAnguage (Mudela), and assesses their suitability for long-term archiving of scanned sheet music. It does not examine every detail of these languages, (they both have adequate reference manuals [HH97] [NNMb],) but instead compares their fundamental concepts. Existing applications that support these languages are also explored.

1 Introduction

It has been argued that Common Music Notation (CMN) is one of the most complex notations in existence [Byr94]. Representing the graphically complex two-dimensional printed score as a one-dimensional string with a very limited character set is such a difficult problem that, in nearly forty years of research, a ubiquitous standard has not emerged. Therefore, compromise is inevitable when selecting a musical representation language. Add to that the concreteness of application and development support, and the compromises are even greater. This document examines some of the shortfalls inherent in GUIDO and Mudela and provides some possible solutions.

1.1 General resources

The book *Beyond MIDI* [SF97b] is an exhaustive survey of musical description languages, (though it predates both GUIDO and Mudela). It also includes chapters regarding issues [SF97a] and guidelines [Hal97] for musical codes. Dannenberg provides an exhaustive overview regarding music representation [Dan93]. As well, there are at least two good indices of musical codes available online [Cas00] [Mou00].

2 Background

2.1 GUIDO

Holger H. Hoos originally developed GUIDO [HHRK98] out of the necessity for a human-readable music description language for the music analysis package *SALIERI* [HH99]. Because of its roots in computational musicology, its core is very clean and free of the messiness of CMN. In partnership with Keith A. Hamel, and others, GUIDO was extended to support features important to CMN.

GUIDO is currently defined in two documents: The Basic and Advanced GUIDO Specifications¹ [HH97] [HHR99]. The forthcoming Extended GUIDO Specification is expected to cover unconventional music notation (eg. microtones, absolute timing, hierarchies.)

2.2 Mudela

Mudela is more closely tied to the practical considerations of typesetting CMN. It originated as the input language to the T_EX-based “music compiler” *LilyPond*, by Han-Wen Nienhuys et al [NNMb], and is now also the official language of the *GNU Music Project*. Its original form borrowed heavily from the Tilia representation used by the *Lime* music editor [CH97] [HB93].

Mudela does not have a formal specification, only a rough user’s guide [NNMb].

3 An introduction to the basic syntax

Both GUIDO and Mudela are encoded as text and are therefore human-readable and portable. While GUIDO syntax is somewhat eclectic, Mudela borrows heavily from T_EX.

3.1 GUIDO

GUIDO is elegantly simple. It defines only three categories of atoms:

3.1.1 Notes and rests

These simple atoms are used only for the most basic elements of music notation: notes and rests. Based on the concept of *representational adequacy*, they are as compact as possible: most redundant information can be left out. For example, when they are omitted, the octave and duration default to the value of the previous note.

¹At the time of this writing, the Advanced GUIDO Specification was not available. For this reason, it is impossible to accurately assess the language. Assumptions about the language are noted throughout this document.

```
d1*3/4 c#-1/8 h&/6 *_2 c2&&/2 cis/4 fa1## sol&0 _8
```

3.1.2 Ordering

The ordering atoms place objects sequentially ([...]) or simultaneously ({ ... }). A minimal GUIDO file must be enclosed in square brackets to indicate that the notes appear in sequential order.

eg. [c1/4 d e f g a b c2/2] % C major scale²



eg. { c e g } % C major triad



3.1.3 Tags

All other elements and properties that modify notes and rests are indicated using tag syntax. Since tags begin with an escape-character (the backslash '\'), it is always possible to distinguish tags from normal note elements. Tags have an *id* (name), zero or more named arguments (in angled brackets '< >') and, optionally, an associated group of notes (in parentheses '()').

```
[ \repeatBegin c4 d e c \repeatEnd ] % Frere Jacques
```



```
\clef<'treble'> % Insert a treble clef  
[ \beam(c8 d e c) ] % Double-time and beamed
```



3.2 Mudela

While Mudela syntax is more complex, it should be natural to those familiar with $\text{T}_{\text{E}}\text{X}$ or $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$. Groupings are formed with curly braces '{ }', and escape commands are preceded with a backslash '\'. Mudela makes extensive use of bracketing: curly braces '{ }' for sequencing, angle brackets '< >' for simultaneity, parentheses '()' for slurs and square brackets '[']' for beams.

Anything between white space can be interpreted as an atom, and packages can be loaded at run time to interpret atoms in new ways. This flexibility makes the language quite convenient for authors, but very convoluted for language

²Due to problems with the freely-available GUIDO tools, all music notation figures are produced with Mudela's *LilyPond* unless otherwise noted.

implementors. It is essentially a moving target: never fully defined without external libraries.

There are four “standard” ways in which atoms are interpreted:

3.2.1 Normal mode

A mode in which atoms have meaning only as strings. The purpose of this mode is unclear.

3.2.2 Note mode

Each atom represents a pitch name. The exact keyword used for pitch names can be changed by loading different language packages. As in GUIDO, redundant information is omitted by passing along octave and duration values from one note to the next.

```
d2. cs8 \times 1/6 {hf8} r2 c'ff2 cs4
```

3.2.3 Chord mode

Chords can be built automatically using notation similar to jazz chord notation. The root of the major chord is followed by any additional, omitted or modified scale degrees.

eg. `c1 c:3- c:9-.5+.7+ c:7^ 5.3 c/e c:7/e`



3.2.4 Lyric mode

Lyrics mode views each atom as a syllable of lyrics followed by a durational value.

eg. `\lyrics Oh4 say, can you see2 by8. the16`



4 Extension framework

If musical information were well understood and fixed, music representation would be a much simpler problem. In reality, we do not know all there is to know and the game is constantly changing. For both of these reasons, it is important that music representations allow extensions to support new concepts and structures.

– Roger B. Dannenberg [Dan93]

Both GUIDO and Mudela have well-defined ways to extend the language.

4.1 GUIDO

GUIDO has an XML-like approach to extensions: new features can be added by using new tags, but the underlying lexical syntax cannot be changed. This allows legacy or domain-specific programs to ignore unrecognized or unimportant tags and utilize the remaining information in a GUIDO file. Since the tag syntax is standardized, it is even possible, in many instances, for an application to manipulate recognized elements while maintaining their relationship to unrecognized tags.

For example, suppose a GUIDO author invented the tag `\editorial` to handle editor's notes.

```
[ b8 e \editorial<'Handel wrote D-sharp'>(d2) e1 ]
```

Now suppose the author wanted to run the file through a program, *transpose*, that is unaware of `\editorial` tags. It might transpose the file up a semi-tone as follows:

```
[ c8 f \editorial<'Handel wrote D-sharp'>(ef2) f1 ]
```

Note that the original editorial comment remains intact, even though *transpose* is unaware of its meaning.³

4.2 Mudela

Like `TEX`, Mudela allows the inclusion of macros to extend the language. Unlike GUIDO's tags, macros can add arbitrary tokens to the language, or even redefine existing ones. In common use are the language packages that change the name of pitches and keywords. For example, if the Mudela author loads the `dutch` package, C-sharp is represented by `cis`, whereas with the `english` package it is `cs`. More complex concepts such as percussion staves are also implemented as macro packages. Packages are written in the Mudela language itself, (or embedded Scheme,) meaning that any extension derived from existing elements can be used by any program that supports Mudela extensions.

5 Human issues

GUIDO and Mudela's different approaches to syntax affect their human-readability⁴.

5.1 Brevity vs. clarity

Mudela's syntactic flexibility is exploited to make the language concise and at times isomorphic (eg. slurs are `()` and accents are `->`). While this makes

³This this does create an odd error: the editorial comment no longer makes sense in the context of the transposed key.

⁴While it is likely that our uses of these languages will be through graphical editors and end users will never see the description code, the human usability of the language may be important in future applications.






	GUIDO	Mudela	
(a) Slurs	<code>\slur(c d e)</code>	<code>c (d) e</code>	
(b) Beams	<code>\beam(c8 d e f g)</code>	<code>[c8 d e f g]</code>	
(c) Artic.	<code>\accent(c) \stacc(d)</code> <code>\ten(e)</code>	<code>c-> d-. e--</code>	
(d) Ties	<code>\tie(c2 c8)</code>	<code>c2 ~ c8</code>	
(e) Dynamics	<code>\intens<'pp'>(c)</code> <code>\cresc(e g)</code> <code>\intens<'ff'>(c2)</code>	<code>c-\pp e \< g</code> <code>\! c-\ff</code>	

Figure 1: Comparison of verbal GUIDO vs. isomorphic Mudela keywords.

the language very convenient for authors, it is somewhat opaque for readers. For instance, it is difficult, without consulting the manual, to determine the meaning of ‘\!’ in Figure 1(d)⁵.

The meaning of GUIDO keywords is more self-evident, at least to English speakers: If one knows the widely accepted English term for a musical element, one already knows the associated GUIDO keyword. This breaks down, however, when keywords are abbreviated. While some abbreviations, such as `\bm`, have equivalent full forms (`\beam`), others do not, eg. `\stacc` (staccato) and `\oct`. By accepting both full and shortened versions for all keywords, the problem of forgetting the correct abbreviations could be eliminated. As it stands, however, GUIDO’s abbreviated keywords are inconsistent.

5.2 Representational adequacy and context-dependence

Both GUIDO and Mudela are based on the concept of *representational adequacy*. This implies that the bare minimum of information is required to represent scores. For example, since notes of a given duration tend to be followed by notes of the same duration, when a duration value is not supplied, it will default to the value of the previous note.

Representational adequacy is a great convenience for authors who are entering scores manually. It can also, in many cases, improve the readability of the text representations because the reader does not have to wade through redundant information. However, it ties the particular representation of phrases to their context (location within the file.) Consider the string of GUIDO notes:

`d e f g`

⁵Answer: It ends the hairpin crescendo.

The appearance of these notes after either `c4/8` or `c2/4` results in an entirely different interpretation. This makes copy-and-paste score writing precarious. The problem is made worse in GUIDO by the absence of macro facilities, necessitating a copy-and-paste style. In Mudela, context dependence is an even bigger mess because atom interpretation is also dependent on the syntax mode.

6 Implementation issues

The context-dependence of representational adequacy is not a problem for computer-based parsing. The score can be normalized (fully specified) on input and distilled (redundancies removed) on output⁶. The core parser and data structures, therefore, do not need to be concerned with representational adequacy.

6.1 Parsing

GUIDO's syntactic simplicity makes parser implementations less complex and more robust. Only a handful of atom types need to be recognized. This simplicity also allows elegant extensions of the language (see Section 4) and handling of undefined tags.

While macros add a lot of power to Mudela files, particularly for logical abstraction (see Section 7), it makes implementation of the language much more difficult. All packages must be available on the host machine. It would be difficult to make a “white room” implementation of a Mudela parser since the exact semantics of Mudela's macro language is undefined outside of the *LilyPond* source code.

7 Logical abstraction

7.1 Logical abstraction in text typesetting

Logical abstraction is commonplace in text typesetting. An author may specify the logical structure of a document, such as chapter or section headings, without specifying any visual attributes. This aids in two things:

1. **Flexible presentation.** The document can easily adapt to different formats. For example, fonts may change if it is used in journals, in books, on large CRT displays or small palmtop devices.
2. **Meaningful searching.** If the different parts of the text are marked by logical meaning, automated searches through a large number of documents can be more accurate. For instance, you may want to search for an author's name only in the “author” field of a number of documents.

⁶The terms ‘normalized’ and ‘distilled’ are my own. I'd be pleased to know if there are standard terms for this process in the literature.

Examples of this paradigm in text formatting appear in the \LaTeX macro package for \TeX [Knu99], style templates in Microsoft Word, the DocBook format [WM99], Cascading Style Sheets in the HTML 4.0 definition [Bos], among others.

7.2 Logical abstraction in music typesetting

Logical abstraction of common music notation is a thornier issue. Donald Byrd's PhD. thesis argues the "non-feasibility of fully-automatic high-quality music notation" [Byr94]. This implies that one cannot leave all issues of visual formatting in the hands of the computer, as is often done in text formatting. For long term archiving of a large number of musical scores, however, it would be advantageous to store as much logical information as possible in addition to the necessary visual formatting.

In most cases, GUIDO and Mudela already specify music logically, using visual overrides only when needed. For example, notes are given as pitch names, with facilities to move notes if the automatic placement is not adequate.

One of the areas where logical specification is weak is with text. In CMN music notation, a number of different types of text are in common use: eg. tempo markings, dynamic expressions, performance instructions, fingerings etc. In most published music, each of these types of text are usually set in a different typeface and in a different position on the staff. The exact typeface and placement varies between publishers and between different types of scores from the same publisher. Another use of logical abstraction in *Lime*'s Tilia representation is to specify which text appears on all extracted parts (eg. tempo markings) and which text appears only with individual instruments (eg. performance instructions). For such things to happen, instead of saying something like "this text 'Allegro' is in 12pt bold," it would be better to say "this text 'Allegro' is in 12pt bold because it is a tempo marking." Using Mudela macros, this would be easy: one could define a macro for tempo markings at the head and use it for all tempo markings throughout the file. In GUIDO, the language could be extended to support tempo markings, but applications with no knowledge of them would not typeset them as text. Instead, one would have to use a dual approach: specifying at both the logical and visual levels.

The advantages of such abstractions cannot be expected to be as great as they are for text. Byrd has argued that fully-automatic flexible presentation would be impossible: changing the typeface of all logical tempo markings would likely break the manual visual layout of the page. Searching would be improved, however: one could perform substring searches only in tempo markings, ignoring all other markings. This would be a great asset to a large music database project.

7.3 Bibliographic information

It would also be convenient to store all bibliographic information related to the score in the file itself. Mudela has a header group in which one can provide information such as title, opus, composer and editor. An extension of this system

is used by the Mutopia Project [Saw], a database of scores in Mudela format, to automatically generate web-based catalogues. The GUIDO specification as it stands does not have such an advanced system⁷. It only defines tags for author and title. However, it would be trivial for one to define tags for this purpose and publish their definitions along side the standard GUIDO specifications. These tags could even be based on an existing bibliographic file standard such as BIBTEX to provide interchange with existing publishing and cataloguing systems.

8 Tools

8.1 GUIDO

8.1.1 Applications

GUIDO NoteServer (Figure 2) [Ren99] displays GUIDO as common music notation inside a web-browser using CGI or Java. At the time of this writing, it supports most of the Basic GUIDO specification⁸. It can be used with any modern web-browser at the Technische Universität Darmstadt website [Ren99].

GUIDO NoteViewer is the stand-alone version of *GUIDO NoteServer* for Microsoft Windows. It is freely available for download, but not open-source. A present, both *NoteServer* and *NoteViewer* are very bug-ridden, producing very innacurate scores, and their primitive layout abilities are useful only for the previewing of very simple GUIDO scores.

Salieri is an algorithmic composition and analysis program that uses GUIDO as its internal data representation [HH99].

NoteAbility is a professional commercial notation program developed by Keith Hamel [Ham99] [Ham98]. *NoteAbility Pro* runs on OpenStep and Mac OS-X and allows fully specified Advanced GUIDO to be imported and exported. The scaled-down *NoteAbility Lite* runs on Microsoft Windows and Mac OS and can export Advanced GUIDO. Importing is limited to Basic GUIDO.

Command-line converters between GUIDO and MIDI also exist.

8.1.2 Development Tools

According to the GUIDO website, last updated in March 1999, the *GUIDO Parser Kit* will be available for free download, though the precise form of the library (i.e. open-source or pre-compiled) is uncertain. The kit has already been used to implement *NoteServer*, *NoteViewer* and *NoteAbility*'s GUIDO import/export abilities [HHR99]. Almost all of the work of normalizing (i.e. fully-specifying) a file, such as the automatic inference of `bar` tags from the `meter`

⁷Bibliographic information may be supported in the unreleased Advanced GUIDO Specification, however.

⁸Informal experience has shown that it is has creates many graphical errors. (See Figure 2.)



Corresponding GUIDO input:

```
{ [\title<"II"> \tempo<"Adagio">
  \staff<1> \clef<"tenor"> \key<-5>
  \meter<"4/4"> \i<"p"> \i<"espress">
  \sl(h&0/8 f1 d&2/4)
  \grace(c/16 d&) \sl(c/8 h&1 c2 d&)
  | \meter<"5/4">
  \sl(h&0/8 f1 d&2/8. c/32 d& e&/8 d&)
  \grace(c/16 d&) \sl(c/8 h&1 c2 \grace(e&) d&)
  ] }
```

Figure 2: Output from *GUIDO NoteServer* displayed in Netscape Navigator. Note that there are many errors in the output, despite the fact that this example was provided by the authors.

tag, is implemented as GUIDO-to-GUIDO transformations, greatly simplifying the task of reading GUIDO files.

8.2 Mudela

8.2.1 Applications

Mudela is the official language of the ambitious *GNU Music Project*, which aims to provide a complete suite of open-source music applications including notation typesetting, sequencing and optical music recognition.

The only *GNU Music Project* application that currently exists in usable form is the music typesetter *LilyPond*. It is a command-line application that converts Mudela input to $\text{T}_{\text{E}}\text{X}$ [Knu99] output which can then be printed or viewed on-screen using standard $\text{T}_{\text{E}}\text{X}$ tools. It currently runs on UNIX and Microsoft Windows, though the Windows port requires large and complex applications that are not commonly installed ($\text{T}_{\text{E}}\text{X}$, Python). Unlike *MusiX $\text{T}_{\text{E}}\text{X}$* [TME99] [Ick97] [Sim00] and *Opus $\text{T}_{\text{E}}\text{X}$* , which are implemented entirely as $\text{T}_{\text{E}}\text{X}$ macros, *LilyPond* is implemented in a combination of C++, Scheme, Python, $\text{T}_{\text{E}}\text{X}$ and the Mudela language itself.

There is also an interactive graphical editor for Mudela, *Denemo* (Figure 3), which is still in the very initial stages of development. Though graphical, the music editing is entirely QWERTY keyboard-based. Interestingly, the keyboard input bears some abstract resemblance to Mudela.

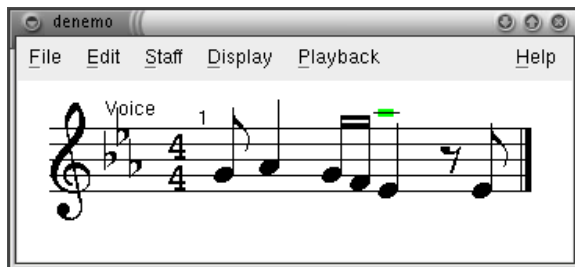


Figure 3: Denemo editor showing first bar of J.S. Bach, “Schlummert ein.”

Keyboard input of Figure 3:

```
g3 a2 g4 f4 e2 Alt-3 3
```

Corresponding Mudela input:

```
g8 af4 g16 f ef4 r8 ef
```

8.2.2 Development Tools

LilyPond's source code is documented such that it could be used as the basis for parsing Mudela. According to the LilyPond internals document [NNMa], its parser may someday be distilled into its own library.

9 Conclusion

For the end user who wants to manually read or edit the language, the choice is perhaps one of personal taste. Mudela's concise syntax can be learned easily and in the informal case study of preparing this document, I have found it to work quite well. In a previous project involving preparing numerous GUIDO scores, I found its verbose syntax cumbersome at times.

The end user is also influenced by the quality of supporting tools. At present time, the freely available GUIDO tools are so buggy as to be useless, leaving the thrifty user or open-source aficionado no choice but to use Mudela with *LilyPond*. I have not evaluated the commercial GUIDO tools from Keith Hamel's Opus One software, but regardless of their design or usefulness, it may be problematic to embark on an open-source project whose only possible interchange is with a commercial product.

From the point-of-view of an implementor, GUIDO is a much more elegant and practical language than Mudela. It is clearly defined and its design ensures a language stability even as new extensions are devised and enter common practice. It therefore represents a relatively small initial and continuing investment from the programmer, especially if the GUIDO Parser Kit is available and works as advertised.

From the available evidence, GUIDO is the best language for long-term archiving of scores. The Advanced GUIDO Specification and the GUIDO Parser Kit will need to be evaluated to ensure that this is in fact the case⁹. They will also be required before any GUIDO development can begin. In an ideal world, GUIDO's superior design should ensure that it receives wide-acceptance. However, in the short term it may be necessary to implement a one-way GUIDO-to-Mudela converter to provide output for users who do not own NoteAblity. Some extensions to GUIDO may also be necessary to fully support bibliographic information and logical abstraction. This may involve adding simple macro support that does not break the existing extension system. I would also like to encourage the inclusion of all full names for the abbreviated tags in the language.

Due to the limited resources of our project (one programmer over a four month period) choosing the simple, elegant solution, GUIDO, seems to make sense, even if the short-term benefits of interoperability may be compromised.

⁹In a recent e-mail correspondence with Holger H. Hoos, he promised to contact me when these were available.

References

- [Bos] Bert Bos. Cascading style sheets. <http://www.w3.org/Style/CSS>.
- [Byr94] Donald Byrd. Music notation software and intelligence. *Computer Music Journal*, 18(1):17–20, 1994.
- [Cas00] Gerd Castan. Common music notation and computers. 2000. http://www.s-line.de/homepages/gerd.castan/compmus/index_e.html.
- [CH97] David Cottle and Lippold Haken. The LIME Tilia representation. In Eleanor Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, 1997.
- [Dan93] Roger B. Dannenberg. Music representation issues, techniques, and systems. *Computer Music Journal*, 17(3):20–30, 1993.
- [Hal97] David Halperin. Afterword: Guidelines for new codes. In Eleanor Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, 1997.
- [Ham98] Keith A. Hamel. NoteAbility - a comprehensive music notation editor. *International Computer Music Conference*, 1998.
- [Ham99] Keith A. Hamel. NoteAbility. Software application, June 1999.
- [HB93] Lippold Haken and Dorothea Blostein. The Tilia music representation: Extensibility, abstraction, and notation contexts for the lime music editor. *Computer Music Journal*, 17(3):43–58, 1993.
- [HH97] Holger H. Hoos and Keith A. Hamel. The GUIDO music notation format: Version 1.0. Technical Report 20/97, Fachbereich Informatik, Technische Universität Darmstadt, 1997. <http://www.informatik.tu-darmstadt.de/AFS/CM/GUIDO/>.
- [HH99] Holger H. Hoos and Thomas Helbich. The SALIERI project. 1999. <http://www.informatik.tu-darmstadt.de/AFS/CM/SALIERI>.
- [HHR99] Holger H. Hoos, Keith A. Hamel, and Kai Renz. Using Advanced GUIDO as a notation interchange format. In *International Computer Music Conference*, pages 395–398, 1999.
- [HHRK98] Holger H. Hoos, Keith A. Hamel, Kai Renz, and Jürgen Kilian. A novel approach for adequately representing score-level music. In *International Computer Music Conference*, pages 451–454, 1998.
- [Ick97] Werner Icking. MuT_EX, MusicT_EX, and MusiXT_EX. In Eleanor Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, 1997.

- [Knu99] Donald Knuth. *Digital Typography*. Center for the Study of Language and Information, Stanford, CA, 1999.
- [Mou00] Steve Mounce. Music encoding standards. 2000. <http://www.student.brad.ac.uk/srmounce/encoding.html>.
- [NNMa] Han-Wen Nienhuys, Jan Nieuwenhuizen, and Adrian Mariano. LilyPond internals. <http://www.cs.uu.nl/people/hanwen/lilypond/>.
- [NNMb] Han-Wen Nienhuys, Jan Nieuwenhuizen, and Adrian Mariano. LilyPond reference manual. <http://www.cs.uu.nl/people/hanwen/lilypond/>.
- [Ren99] Kai Renz. GUIDO NoteViewer. Software application, August 1999.
- [Saw] Chris Sawyer. Mutopia project. <http://www.uwaterloo.ca/mutopia/>.
- [SF97a] Eleanor Selfridge-Field. Beyond codes: Issues in musical representation. In Eleanor Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, 1997.
- [SF97b] Eleanor Selfridge-Field, editor. *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, 1997.
- [Sim00] Don Simons. PMX: A preprocessor for MusiX_{TEX}. Software application, 2000. <http://www.gmd.de/Misc/Music/>.
- [TME99] Daniel Taupin, Ross Mitchell, and Andreas Egler. MusiX_{TEX}: Using _{TEX} to write polyphonic or instrumental music. 1999. <http://www.gmd.de/Misc/Music/>.
- [WM99] Norman Walsh and Leonard Mueller. *DocBook: The Official Guide*. O'Reilly and Associates, Sebastopol, CA, 1999. <http://www.docbook.org/>.