An editor for lute tablature

Christophe Rhodes and David Lewis

Centre for Cognition, Computation and Culture
Goldsmiths College, University of London
New Cross Gate, London SE14 6NW, UK
c.rhodes@gold.ac.uk, d.lewis@gold.ac.uk

Abstract. We describe a system for the entry and editing of music in lute tablature. The editor provides instant visual and MIDI feedback, mouse and keyboard controls, a macro recording facility, and full runtime extensibility. We conclude by discussing planned future functionality and considering other potential applications for the technology.

1 Introduction

The Electronic Corpus of Lute Music[1] seeks to act as a first point of reference for researchers in lute music and to raise the profile of the repertoire. The Western European lute is an instrument of great historical importance, and Ness estimates [1] a still extant repertory of nearly 60,000 pieces scored for the instrument. Yet, despite its clear significance, the lute and its music play a comparatively minor part in current musicology: the music is not generally well-known and its historical role rarely discussed.

This obscurity arises in part because the surviving sources are often miscellanies, making location of works by a single composer or finding sources for a specific piece difficult. Additionally, the notations for lute music are very different from staff notation: anyone curious to explore the repertory must first learn the notation and, until they become experienced, to transcribe the pieces into a more familiar format.

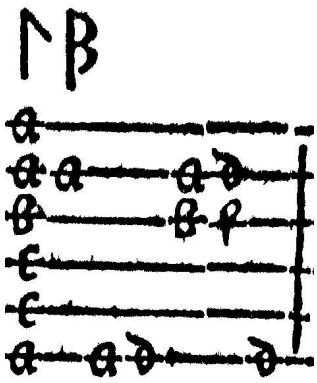
ECOLM [2] seeks to make it easier for lute music to be appreciated by non-experts, without the need to understand either tablature or the technique of the instrument, whilst still providing scholars with the detailed, specalist information where it is required (see [3] for an example). The core of the project consists of musical encodings, forming an online edition, providing 'diplomatic facsimiles' (i.e. literal transcriptions), editions, and computer-generated MIDI and staff transcriptions. We also seek to create an infrastructure for distributed editing, necessitating an encoding system and a user interface.

[1] ECOLM 2 is the second phase of a five-year grant provided by the UK Arts and Humanities Research Board (now Council)

2 Lute Tablature and *TabCode*

Lute music is written in a variety of different tablature forms. The notation tells the lutenist which strings should be played at which frets and when. The most common approach is to use horizontal lines (similar to staff lines) to represent strings, and letters or numbers placed on them to represent frets – a practice that survives to this day in guitar tablature. The music is read left to right, with rhythm signs placed above to indicate timing. At its most basic, lute tablature is entirely sequential in nature, with one chord following another without notated overlap, and it is impossible to indicate multiple simultaneous rhythms.

Clearly, this attribute makes the transfer of the notational information into ASCII for use in the corpus a much simpler proposition than has been the case for the majority of classical music scores. Crawford [4] describes a format called *TabCode* for encoding lute tablature as a series of 'tabwords' separated by white space. Each word begins with a character indicating the rhythm sign (the initial of the name of that sign: `H` for half note, `Q` for quarter, etc.), if present, followed by a letter-number pair for each symbol in the chord, with the number signifying the string and the letter the fret.



```
Qa1a2b3c4c5a6
Ea2
a6
d6
a2b3
d2f3
d6
|
```

Fig. 1. An extract from 'Fantasia Ioannis Dooland Angli *Lachrimae*', Jean-Baptiste Besard, *Thesaurus Harmonicus* (1603), f.16v, and its *TabCode* encoding.

In order to create camera-ready artwork, in particular for [5], Tim Crawford wrote The Tablature Processor, a Macintosh-based application with its own binary file format, but capable of importing and exporting *TabCode*. The Tab Processor has some facility for data entry, but is primarily a type-setting program for a type of lute tablature.

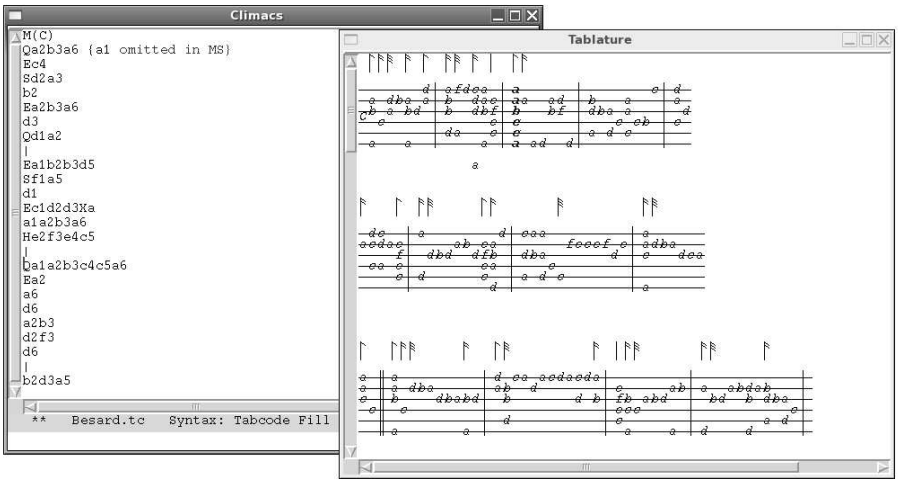


Fig. 2. The tablature editor and graphical display. Note the third bar of the tablature, which corresponds to the fragment in figure 1.

3 TabEditor

We have developed a new application to speed the process of entering and editing *TabCode*. This application provides a syntax-aware editor in the Emacs tradition, along with a graphical view of the currently-edited score, which updates in real-time as the user manipulates the textual *TabCode* buffer. In addition, this graphical view is mouse-sensitive (as is the editor buffer) and allows interaction with the graphical objects, while retaining the primacy of the text representation: manipulation of the graphical object is implemented as a sequence of manipulations on the text. We also allow the option for the user to receive immediate audio feedback of the current chord at its completion, as well as region or full-piece audio rendering.

The major improvement in this application compared with previous editors is the immediacy of the feedback; however, from the editor's Emacs heritage comes extensibility, both through explicit definition of additional functionality at run-time and through the ability to record keyboard macros for automation of repetitive tasks.

This application is not at present able to produce an edition-ready rendering of *TabCode*, unlike the Tablature Processor; this stems directly from the fact that the textual *TabCode* is required to contain all the information in the application: the language is rich enough to express the semantics of a lute tablature manuscript, but not the necessary tweaks that an editor makes for the print copy of a work.

3.1 Implementation Details

The application is written in Common Lisp, using SBCL, the McCLIM [6] implementation of the CLIM specification for graphical interface management, and Climacs, a CLIM editor, as our editor substrate. It is beyond the scope of this paper to give a detailed exposition of CLIM's capabilities: we refer the interested reader to other sources such as [7]. For our purposes, CLIM associates graphical output with application data through *presentations*, and manages the efficient redrawing of application state through *incremental redisplay*.

The CLIM presentation facility provides what is in some sense an object-oriented renderer: when drawing the graphical view of the tablature denoted by the *TabCode*, the graphical elements retain their association with the `tabword` objects resulting from parsing the editor buffer. This then allows a trivial implementation of actions such as moving the cursor to the point in the buffer corresponding to a particular chord in the graphical view, and a relatively simple implementation of commands for musical manipulation (such as one to move glyphs up a string to correct a typographical error) operating on the textual *TabCode* but trigged by an action in the graphical view.

Although the Climacs editor includes a syntax analysis module [8] based on a parser implementing Earley's algorithm [9], the generality of this framework was not needed for a language as simple as *TabCode*. Instead, we implemented a combined lexer and parser, which on a parse error preserves the partial parse, if any; advances to the nearest lexically following whitespace; and resets the analyser's state. This parser generates a sequence of `tabwords` from the text in the editor buffer, reusing portions of the previously-generated sequence if it can prove, based on the extent of the text region 'damaged' by user interaction, that the parse is unchanged.

Incremental redisplay is in some sense merely an optimization, but it is a sufficiently broad one that it merits discussion: it permits the system to avoid redrawing output if it can determine that this will not be necessary, on an object-by-object basis. We use this optimization by preserving the identity of those elements of the parsed buffer contents which can be proved not to have changed, as described above. This cache not only informs the display within the editor buffer – highlighting parse errors, for example – but also the graphical view: a chord need not be redrawn if it has not changed since the last edit.[2]

4 Conclusions and Future Directions

The real-time feedback provided by this application has met with approval from its users, including some with limited technical skills: errors are corrected more quickly, and the ability to find the area in the *TabCode* source corresponding to a place in the graphical output allows more efficient navigation. The CLIM

[2] In addition to this, it should also not have changed its position; this is the case even for many edits in a *TabCode* document: only those which change an element's width will affect the positioning of subsequent elements on that line.

framework assisted us in development of this application by providing the means to associate high-level application data directly with the graphical output.

TabCode is a simple language when interpreted as a sequence of tabwords representing notated elements. However, for general semantic analysis and display purposes, a slightly more sophisticated parsing framework is required: to accommodate hierarchical groupings such as beaming, connecting lines and so on tabword groups must be formed. These hierarchical groups can be incrementally maintained in the same manner as the sequence of tabwords, by computing the overlap of groups with the 'damaged' region in the editor buffer.

A feature planned for the near future is transcription of the current buffer to score (in Common Music Notation). The initial transcription algorithm should prove fairly simple, as each chord can simply be directly transcribed; difficulties remain in the areas of pitch spelling and readable polyphony.

We believe it would be relatively simple to adapt our application to support other textual representations of music (such as Humdrum's [10] `**kern`) or more generally of two-dimensional data, while maintaining the association between graphical display and textual input.

Acknowledgments

C.R. and D.L are supported by EPSRC grant GR/S84750/01 and AHRC grant B/RE/AN9717/APN15559 respectively.

References

1. Ness, A.J., Kolczynski, C.A.: Sources of lute music. In Sadie, S., Tyrrell, J., eds.: The New Grove Dictionary of Music and Musicians. Volume 23. Macmillan, London (2001) 39–63
2. Crawford, T., Gale, M., Lewis, D.: An Electronic Corpus of Lute Music (ECOLM): technological challenges and musicological possibilities. In Parncutt, R., ed.: Conference on Interdisciplinary Musicology, Graz (2004) 118–119
3. Lewis, D., Gale, M.: "La battaglia": a computer-assisted approach to an extended musical family. Presented at the Annual Conference of the Renaissance Society of America (2005)
4. Crawford, T.: Applications Involving Tablatures: *TabCode* for Lute Repertories. Computing in Musicology **7** (1991) 57–59
5. Crawford, T., ed.: Silvius Leopold Weiss: Sämtliche Werke für Laute. Volume 5–7. Bärenreiter, Kassel (2002–)
6. Strandh, R., Moore, T.: A Free Implementation of CLIM. In: International Lisp Conference, San Francisco, Franz Inc. (2002)
7. Rao, R., York, W.M., Doughty, D.: A guided tour of the Common Lisp interface manager. ACM SIGPLAN Lisp Pointers **4** (1990) 17–37
8. Rhodes, C., Strandh, R., Mastenbrook, B.: Syntax Analysis in the Climacs Text Editor. In: International Lisp Conference, Stanford (2005) (accepted for publication).
9. Earley, J.: An Efficient Context-Free Parsing Algorithm. Communications of the ACM **13** (1970) 94–102
10. Huron, D.B.: The Humdrum Toolkit. CCRAH, California. (1994)