PIECE: A Music Language Editor Synchronized with Graphical Views

Satoshi Nishimura
School of Computer Science and Engineering
The University of Aizu
Aizu-Wakamatsu city, Fukushima, 965-8580, Japan
Email: nisim@u-aizu.ac.jp

Abstract

This paper describes a music editing system called the PMML Integrated Emacs-based Composing Environment (PIECE). The system provides both a textual-language view and piano-roll view synchronized with each other. The consistency between the views is maintained even if control structures or macros are used in the language texts. A novel algorithm for maintaining the consistency is presented.

1 Introduction

In MIDI sequencers, there are two forms of music representation through which music is edited. One is graphical representation such as piano rolls or conventional music notations. The advantage of graphical representation lies in intuitiveness; one can easily understand the correspondence between actual music and the representation. On the other hand, graphical representation is weak in structured or algorithmic description. The other form is textual representation such as music languages. Although textual representation is inferior in intuitiveness, it is the most suitable for structured/algorithmic description.

To make use of all the advantages, this paper treats with a music editing system supporting both of the forms. The system contains both a text editor specialized for a music language and a graphical editor based on a piano roll which is unstructured, time-linear representation of music (Figure 1). As the music language, the PMML [1] is employed; however, some of techniques described in this paper are also applicable to other languages. The text editor is the GNU Emacs editor extended for the PMML, through which users can edit the text in a usual way. On the piano-roll editor, users can modify pitch, onset time, duration, velocity, and control values by dragging “note bars” or “value dots” with pointing devices.

In the above system, the synchronization between the textual and graphical views is maintained. For example, clicking a note bar in the piano roll moves the cursor to the corresponding position in the text window. Conversely, issuing a synchronization command at some location in the text window scrolls the piano roll to the corresponding position. In addition, modification in the language text is reflected in the piano roll, and vice versa. For example, when the text is changed and then the synchronization command is issued, the piano roll is updated correctly. Conversely, as illustrated in Figure 1, dragging a note bar or value dot in the piano roll adjusts the text appropriately.

There are many commercial MIDI sequencers supporting the synchronization of graphical views and textual views [2]; however, textual views in those systems are simple time-linear lists of events. The system in this paper differs in that control structures or macros can be used in the texts, and as a result, it enables us to modify note attributes or control values from the graphical view with keeping the algorithmic structures of music.

![Figure 1: Music editor with synchronized views](image)

2 Features of the PMML

The PMML is a music description/manipulation language designed for MIDI sequencing. It supports both direct music description and algorithmic representation.

The PMML has a feature for specifying the attribute values of a particular note relatively to their default values in addition to specifying them absolutely. This feature is very useful in reflecting changes in the graphical view to the text. For example, if the velocity of the note described by
There are two sources of such inconsistency. One is one-to-many relationships between an event-generating command in the text and symbols in the piano roll. An example for this is shown in Figure 2(a). Due to the loop structure, each note command has two corresponding bars in the piano roll, and therefore, dragging one bar requires not only updating the command text but also moving the other bar to maintain consistency. The other source of the inconsistency is subordinated notes. In Figure 2(b), because the effector `add_notes(n+12)` is attached, each note accompanies a subordinated note whose pitch is an octave higher. Also in this case, dragging one note bar should move the other in parallel.

Effectors which transform note attributes will cause another type of difficulty, as exemplified in Figure 2(c). The effector `modify_notes(v+1)` will scale the velocity of each note by two. Therefore, when the velocity of the note is increased by 20 from the graphical view, the correct way to adjust the text is increasing the velocity by 10.

The above last example requires the inversion of the transformation process, and of course, it is impossible to handle a general case. In our system, we treat with only the transformations described by a linear expression $ax + b$ ($a \neq 0$). For other transformations, modification from the graphical view with respect to that attribute is inhibited.

To identify which graphical symbol should be updated and how the text is adjusted, we introduce a directed graph $G = (V, E)$ called an event dependency graph. Each vertex of $G$ is an event vertex, S-vertex (synchronization vertex), or U-vertex.
for(i=1; i<2; i++) { // repeat 2 times
    // set the default velocity
    // (first time: 40, second time: 50)
    v = ($i == 1) ? 40 : 50
    // play a note with a grace note
    // with letting the velocity of the
    // grace note to be the half of the
    // original one
    C(grace(D(vv=0.5)))
}

// attach an effect for clipping velocity
modify_notes(vw (vw=80) ? 80 : v)
E // play an E note

Figure 4: An example containing grace notes

velocity before editing  
/ velocity after editing

Figure 5: Event dependency graph corresponding to Figure 4

determine W as Vertices (1) and (2). If Vertex (1) is chosen in Step 2, the velocity values of Vertices (1) through (4) will be changed as indicated in Figure 5. In this case, the resultant text will be 'C(grace(D(vw=0.5))) (vw=20)'. If Vertex (2) is taken in Step 2, only the velocity of Vertices (2) and (4) will be changed, and the resultant text will be 'C(grace(D(vw=0.5) (vw=10))'. In either case, the synchronization between the two views are correctly maintained. Meanwhile, the modification of the E note from the graphical view will be inhibited.

4 Implementation

The PIECE system is implemented using three processes of the Unix operating system (Figure 6). Currently, the system runs under the Linux or SGI/IRIX operation system. The Emacs process is the GNU Emacs editor extended with Emacs Lisp codes. The MIDI server process, developed with the C language, is responsible for MIDI input/output and time management. The graphics editor process, developed with the Java language, displays the piano-roll view and handles modification requests from the user. These three processes are connected by pipes, which transmit controlling information such as playing start/stop requests, po-
positioning requests, text-highlighting requests, and text-adjustment requests.

MIDI event data as well as information about source positions and event dependencies are provided to the graphical editor and MIDI server processes via an extended MIDI file. The extended MIDI file is created by the PMML compiler upon a request from the Emacs process.

The structure of the extended MIDI file is illustrated in Figure 7. To include additional information in the Standard MIDI File with keeping compatibility, we appended non-standard data chunks at the bottom of the Standard MIDI File. The string table chunk contains the strings of macro and file names. The file table chunk specifies the mapping between source file IDs and file names. The macro node chunk contains a tree-formed data structure representing the macro calling sequence to each event-generating command. This information enables the identification of every related position in the text when a note bar is clicked in the piano roll. The event node chunks encode the information of the event dependency graph, in which the S-vertices are implicitly expressed by that two or more events have the same source position. The total size of the extended MIDI file is typically twice as large as the Standard MIDI file.

Figure 8 shows a screen snapshot of the running system. When the user issues a synchronization command in Emacs, the Emacs process updates the extended MIDI file using the PMML compiler, and then sends a positioning request to the graphical editor process so that the piano roll scrolls to the position corresponding to the text cursor location. When an event is dragged in the piano-roll view, the graphics editor process executes the event updating algorithm described in Section 3, and sends a text-adjustment request to the Emacs process, which then does necessary text updates.

5 Summary

The paper described a newly-developed music editing system named PIECE. By providing both a textual-language view and piano-roll view synchronized with each other, the system enables the structured or algorithmic description of music without sacrificing intuitiveness. In the future, we would like to add the synchronized views of conventional music notations into the system.

References