Proceedings of the SMC 2009 - 6th Sound and Music Computing Conference, 23-25 July 2009, Porto - Portugal

SCORE PLAYBACK DEVICES IN PWGL

Mikael Laurson
CMT, Sibelius Academy
laurson@siba.fi

Mika Kuuskankare
CMT, Sibelius Academy
mkuuskan@siba.fi

ABSTRACT

This paper presents a novel system that allows the user to customize playback facilities in our computer-assisted environment, PWGL. The scheme is based on a class hierarchy. The behavior of an abstract root playback class containing a set of methods can be customized through inheritance. This procedure is demonstrated by a subclass that is capable of playing MIDI data. This playback device allows to realize automatically multi-instrument and micro-tonal scores by using pitchbend setups and channel mappings. Also continuous control information can be given in a score by adding dynamics markings and/or special Score-BPF expressions containing break-point functions. We give several complete code examples that demonstrate how the user could further change the playback behavior. We start with a simple playback device that allows to override channel information. Next we discuss how to implement the popular keyswitch mechanism in our system. This playback device is capable of mapping high-level score information with commercial orchestral database supporting keyswitch instruments. Our final example shows how to override the default MIDI output and delegate the play events to an external synthesizer using OSC.

1 INTRODUCTION

Computer-assisted composition (CAC) systems ([1], [2], [3]) have not focused on advanced playback facilities. Users can typically audition scores and other musical raw-material through basic MIDI playback routines that support simple note-on, note-off, pitch and velocity data. A notable exception of this rule are special cases to handle micro-tonal playback which is not well supported by the MIDI standard.

Commercial notation software packages (Sibelius, Finale, Igor), by contrast, do support tools that allow the user to audition orchestral scores. With the advent of recent high-quality orchestral sample databases (EastWest, Vienna Instruments, Garritan) and/or instrument synthesizers (Wallander Instruments, Synful) orchestral simulations are getting more and more convincing. Thus notation software systems combined with orchestral playback facilities have quickly become everyday tools for composers and arrangers not only in the film industry but also for musicians belonging to the contemporary music genre.

In this paper we investigate possibilities that would allow the user to combine various playback options, such as micro-interval playing and orchestral simulation in a CAC environment. Also the output would not be bound to a given orchestral database (as is the case in several notation systems where the samples are bundled with the application). The user should be able to customize playing routines for different libraries and synthesizers, and the control output does not necessarily have to be MIDI-oriented.

PWGL has a long history in controlling physics-based instruments [4] using our notation package ENP [5]. This research has resulted in several tools that allow to enrich basic score information, such as performance rules, scripting, tempo functions, and an enhanced set of expressions that can be inserted in the score either algorithmically or by hand. Other special extensions, such as the macro-note scheme [6], allow the user to further modify and enrich basic score information. Sound examples can be found at: www.siba.fi/pwgl/pwglsynth.html.

In the following we will concentrate on a novel extension of the PWGL system that allows the user to define the behavior of the playback engine. Each time the user starts to play a score the current playback device is evoked. PWGL contains a library of predefined playback devices. This library is written in Common Lisp and CLOS and it can be extended by subclassing one of the existing playback device classes. Multiple inheritance can also be used to combine features from several superclasses in the system. All playback devices support a standard protocol having four main steps: (1) the system first calls an initial preparation method, (2) then it collects the actual playback information, (3) next a setup method is called, and finally (4) the collected data is sent to the current output device.

The rest of the paper is organized as follows. First we give some background information concerning the general playback device scheme in PWGL. Then we discuss in more detail the current status, and enumerate which devices are already present in the system. After this we describe case studies that will show how the user can override the default behavior in order to customize her/his needs for auditioning

SMC 2009, July 23-25, Porto, Portugal

of musical scores.

2 GENERIC PLAYBACK DEVICE CLASS AND ITS METHODS

Our playback system is based on a hierarchy of CLOS classes. At the root we have a generic class called 'pwgl-playback-device'. This class definition contains a set of primary methods to support the four main steps used by the playback scheme. Typically, the user should not override the primary methods (although this is also possible), but should instead redefine the secondary methods. The secondary methods have identical names as the primary ones except for an extension '*' at the end of the name. Next we enumerate the most commonly used methods in the playback scheme.

The method 'prepare-playback' is used to prepare score playback before event calculation. This method can be used to open a sample player application, load sound samples, prepare an instrument setup, etc.

After this initial phase the system starts to collect play data. By default it checks whether the current score has a selection or not (note that ENP also supports discontinuous selections). If a selection is found then only those notes that belong to the selection are considered, otherwise all notes are collected.

After this the system calls for each collected note the methods 'add-playback-cc-events' and 'add-playback-note-event'. The first one is used to collect continuous control information, and the second one is used for note events. (Before the 'add-playback-note-event' also the special method 'add-playback-note-pre-event' is called; we will come back to this method later in this paper in Section 5.) These methods are similar as they should at the end collect association lists (a list containing keyword/data pairs) that are meaningful for the current playback output. For instance if the play information is sent to MIDI then a note event list should contain information dealing with bus (port), status, key and velocity. Internally the 'add-playback-note-event' method calls the 'calc-playback-event' method that builds the association list based on the data that is returned from the 'calc-playback-chan/midi' and 'calc-playback-vel' methods.

Next, the system calls the 'setup-playback' method that is called after event calculation and before playback. This method can be used, for instance, to send volume and pitch-bend information just before playback.

Finally, the realtime playback starts and for each event data list the system calls 'send-playback-event'. Normally the method simply utilizes the pre-calculated association lists and sends the appropriate information to the current playback output.

3 THE DEFAULT PLAY DEVICE

As such 'pwgl-playback-device' is an abstract class and should be subclassed in order to be functional. In this section we outline a typical subclass that is specialized for MIDI output. This class is called 'midi-playback-device', and it is in fact the default play device in PWGL. 'midi-playback-device' has several options for output. First, the final output can either be a realtime stream of MIDI events or a MIDI file. Second, PWGL supports also a special mode for Mac OS X users where play data can be sent directly to a QuickTime synthesizer.

3.1 Micro-tonal playing

'midi-playback-device' supports up to eight MIDI buses or ports (in PWGL a note can have a channel number ranging from 1 to 128, where by default channels from 1 to 16 are sent to port 1, channels 17-32 to port 2, and so on).

During the setup phase 'midi-playback-device' calls the 'setup-playback' method that has two main tasks. First, the user has an option to send a default MIDI volume value to each channel. Second, 'setup-playback' sends pitch-bend data to detune channels in order to support micro-tonal playing. This is done by first analyzing the pitch information of the current score. If the score requires quarter-tone tuning then even-numbered channels are detuned by 50 cents; in case of eighth-tone tuning cannel 2 is detuned by 50 cents, channel 3 by 25 cents and channel 4 by 75 cents, and so on. Based on the micro-tonal content the mapping of MIDI channels is done as follows: if the score is in equal temperament (i.e. there are no micro-tones), then note channel numbers will not change, i.e. channel 1 is played on channel 1, and so on; if the score requires quarter-tone resolution then notes with channel number 1 are delegated either to channel 1 or 2 depending on the pitch content, notes with channel 2 are delegated to channel 3 and 4, and so on; in case of eighth-tone resolution notes with channel number 1 are delegated to channel 1, 2, 3 and 4 depending on the pitch content, and so on.

This scheme with detuning and channel delegation is transparent to the user and it allows to play micro-tonal and multi-instrument scores automatically. As the system supports 128 channels we have a theoretical upper limit of 1/256-tone resolution for micro-tonal tuning. If the user needs multi-instrument scores, this limit is of course lowered as each instrument needs a dedicated set of channels in a micro-tonal context.

During the event data collection phase 'midi-playback-device' calls the method 'calc-playback-event' that performs MIDI port, channel and pitch mapping as described above and returns an association list of raw MIDI data that is optimized for realtime playing. Finally, the 'send-playback-event' method utilizes this information and calls in realtime

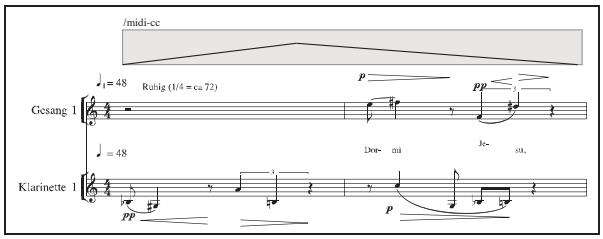

Figure 1. Continuous control options in ENP: (1) crescendo and diminuendo expressions, (2) Score-BPF expression containing a break-point function.

the low-level MIDI event routines in order to output the final MIDI data.

3.2 Continuous control

'midi-playback-device' supports also MIDI continuous control information. The PWGL preference pane has an option that allows to convert automatically ENP expressions to a stream of continuous control MIDI events. Figure 1 shows a two-part score that has several interleaved crescendo and diminuendo markings. When a dynamics expression is encountered in the score during the event calculations phase the 'add-playback-cc-events' method will be evoked. There are two basic options how the dynamics expressions are interpreted: (1) the system creates automatically a ramp that is sampled to get discrete MIDI continuous control events; or (2) the expression contains internally a break-point function that has been edited by the user (ENP allows to open the expression and edit the internal breakpoint function with the mouse). Continuous control information playback is also supported by the special Score-BPF expression (see in Figure 1 the Score-BPF expression above the first staff with the label 'midi-cc'). A Score-BPF can have up to three break-point functions each with individual continuous control designations.

4 CHANNEL PLAYBACK DEVICE

In the following sections we discuss various case studies where we change the behavior of the default PWGL play device. For each case we give a class definition followed by a redefinition of one of the methods discussed in Section 2. Finally we call 'add-playback-device' in order to add the new device to the PWGL playback device library.

We start with a simple case where the idea is to disregard the standard channel information. Instead we check the 'instrument' slot of the note whether it is found in a list of woodwind and brass instruments. Working with symbolic instrument names in conjunction with our notation tools is often more convenient than working with abstract channel numbers (see Figure 2). We use the Lisp function 'position'

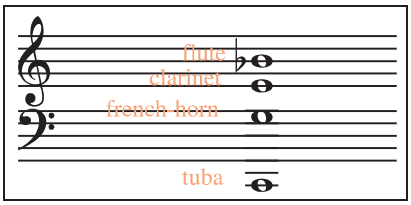

Figure 2. A chord where each note has a different instrument assignment.

to calculate the channel number (thus here 'flute' will be mapped with channel 1, 'oboe' with 2, etc.).

```
(defclass channel-player (midi-playback-device) ())

(defmethod calc-playback-chan/midi*
    ((device channel-player ) (note note))
  (let ((chan (position (instrument note)
          '(:flute :oboe :clarinet :bassoon :trumpet
            :french-horn :trombone :tuba))))
    (values (if chan (1+ chan) (chan note)) (midi note))))

(add-playback-device 'channel-player "channel-player")
```

5 ENP EXPRESSIONS AND KEYSWITCH EVENTS

Next we discuss a generic playback class, called 'keyswitch-player', that is useful when working in conjunction with orchestral databases. 'keyswitch-player' is a subclass of 'midi-playback-device' and thus inherits all features from that class. The idea is to add support to the keyswitch mechanism supported by several commercial sample databases. A keyswitch event is an additional note-on event where the key value is outside the normal range of an instrument. The keyswitch event is normally sent just before the actual note-on event. This allows to instruct the sample database application which specific articulation should be used for the next normal note event. Thus, typically, a keyswitch event having the low C0 as key value for a flute part could mean 'legato', while D0 could mean 'staccato', and so on.

The keyswitch mechanism is straightforward to implement in our scheme due to the 'add-playback-note-pre-event' method that is called just before the actual note event calculation (Section 2). Furthermore, it fits nicely in our system as ENP supports a wide range of expressions (see Figure 3). 'keyswitch-player' adds two new generic methods to the system called 'ksw-expression-no' and 'ksw-offset-no' that should be refined by a subclass of 'keyswitch-player'. These methods are used to calculate the final key value needed for the keyswitch event.

Next we discuss a concrete problem how we can map an instrumental score with expressions denoting various playing techniques and articulations with a keyswitch instrument found in the commercial EastWest sample library. For this end we define a subclass of the 'keyswitch-player' class called 'ew-player '. We need next to associate a keyswitch

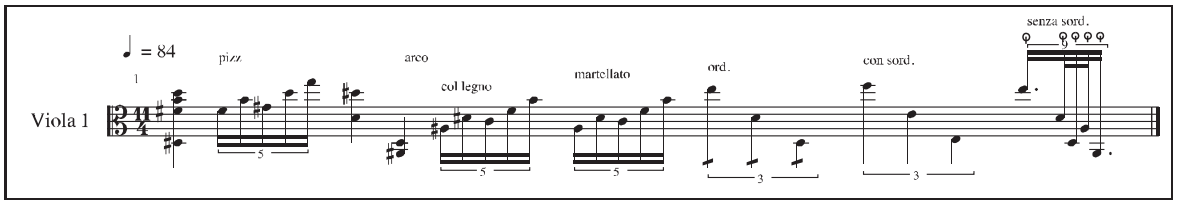

Figure 3. An ENP score for solo viola. The expressions are automatically mapped with correct keyswitch events.

event with an instrument (here viola) and various expressions (see Figure 3). This is done by defining several versions of the 'ksw-exp-no ' method that are specialized for each available expression name ('pizz', 'col-legno, 'martellato', etc.). Each method returns a unique number that– when added to the offset number returned by the 'ksw-offset-no' method–is used by the keyswitch event.

```
(defclass ew-player (keyswitch-player) ())

(defmethod ksw-offset-no
  ((device ew-player) (ins viola)) 24)

(defmethod ksw-exp-no
  (device ew-player) (ins viola) exp) 0)
(defmethod ksw-exp-no
  ((device ew-player) (ins viola) (exp pizz)) 1)
(defmethod ksw-exp-no
  ((device ew-player) (ins viola) (exp col-legno)) 2)
(defmethod ksw-exp-no
  ((device ew-player) (ins viola) (exp martellato)) 3)
(defmethod ksw-exp-no
  ((device ew-player) (ins viola) (exp tremolo8)) 4)
(defmethod ksw-exp-no
  ((device ew-player) (ins viola) (exp bartok-pizzicato)) 5)
(defmethod ksw-exp-no
   ((device ew-player) (ins viola) (exp con-sordino)) 6)

(add-playback-device 'ew-KSW-player "ew-KSW-player")
```

As the ENP instrument database is also based on a CLOS class hierarchy (thus 'viola' is a subclass of 'bowed-strings') the code above can easily be modified so that the keyswitch mechanism is valid for all bowed string instruments, for instance:

```
(defmethod ksw-exp-no
  ((device ew-player) (ins bowed-strings) (exp pizz)) 1)
```

6 OSC PLAYBACK DEVICE

Our final example demonstrates how we can define a playback device that is not using MIDI as output. This is done by defining a subclass of 'midi-playback-device' called 'OSC-playback-device'. Here we want to send OSC [7] messages to an external synthesizer (in this specific case to SuperCollider, [8]). For this reason we redefine the 'send-midi-event*' method. Now, instead of calling the standard MIDI routines, we use the association list information ('midi-info') collected by the event collector and convert the channel, the midi and the velocity data to OSC messages.

```
(defclass OSC-playback-device (midi-playback-device) ())

(defmethod send-midi-event
  ((device OSC-playback-device) midi-info &optional vel?)
  (when-let (osc-stream (read-key :osc-stream))
    (let ((chan (read-key midi-info :chan))
          (midi (read-key midi-info :midi))
          (vel (read-key midi-info :vel)))
      (if (zerop vel)
          (cl-osc:write-osc-message
           osc-stream nil "/noteOff" midi chan)
        (cl-osc:write-osc-message
         osc-stream nil "/noteOn" midi vel chan)))))

(add-playback-device 'OSC-playback-device "osc")
```

7 CONCLUSIONS

The paper presents a playback scheme that allows to convert high-level score information to event lists. The events can be sent to MIDI or to external synthesizers using, for example, the OSC protocol. The system is programmable and through inheritance device subclasses can change the standard behavior of the default playback device of PWGL. This scheme offers many interesting applications such as allowing ENP scores to control orchestral databases. The new playback protocol, when combined with the other tools found in PWGL (i.e. performance rules, scripting, tempo functions, macro-notes), forms a unique system for score-based playback control.

8 ACKNOWLEDGMENTS

The work of Mikael Laurson and Mika Kuuskankare has been supported by the Academy of Finland (SA 114116 and SA 105557).

9 REFERENCES

[1] Laurson, M., *PATCHWORK: A Visual Programming Language and some Musical Applications*. Studia musica no.6, doctoral dissertation, Sibelius Academy, Helsinki, 1996.

[2] Assayag, G., C. Rueda, M. Laurson, C. Agon, and O. Delerue, "Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic", *Computer Music Journal*, vol. 23, pp. 59–72, Fall 1999.

[3] Laurson, M., M. Kuuskankare, and V. Norilo, "An Overview of PWGL, a Visual Programming Environment for Music", *Computer Music Journal*, vol. 33, no. 1, 2009.

[4] Laurson, M., V. Norilo, and M. Kuuskankare, "PWGLSynth: A Visual Synthesis Language for Virtual Instrument Design and Control", *Computer Music Journal*, vol. 29, pp. 29–41, Fall 2005.

[5] Kuuskankare, M. and M. Laurson, "Expressive Notation Package", *Computer Music Journal*, vol. 30, no. 4, pp. 67–79, 2006.

[6] Laurson, M. and M. Kuuskankare, "Towards Idiomatic and Flexible Score-based Gestural Control with a Scripting Language", *Proceedings of NIME'08 Conference*, (Genova, Italy), pp. 34–37, 2008.

[7] Wright, M., "Open sound control: an enabling technology for musical networking", *Organised Sound*, vol. 10, pp. 193–200, 2005.

[8] McCartney, J., "Continued Evolution of the SuperCollider Real Time Environment", *Proceedings of ICMC'98 Conference*, pp. 133–136, 1998.