From Score-Based Approach Towards Real-Time Control in PWGLSynth

Mikael Laurson*, Vesa Norilo*, and Henri Penttinen†
* Centre for Music and Technology, Sibelius Academy,
  laurson@siba.fi, vnorilo@siba.fi
† Laboratory of Acoustics and Audio Signal Processing, Helsinki University of Technology,
  Henri.Penttinen@hut.fi

Abstract
This paper presents a system that combines score-based and real-time control of sound synthesis. The aim is to extend the current score-based approach that allows to produce complex control information out of our music notation package. This complementary real-time extension is attractive as it gives an ‘instrument-like’ feedback for the user. Instruments can be calibrated interactively, and different texture types can be tested in a more flexible way than with the score-based approach only. After some background information we discuss how the current score-based system is extended to cover real-time applications. We give also some concrete real-time instrument model examples that demonstrate some of the potentials of the system. The synthesis algorithms used are physics based sound synthesis models.

1 Introduction
One main focus in our research has been to develop an interface between music notation and sound synthesis (Laurson, Norilo, and Kuuskankare 2005) using our Lisp-based visual programming environment called PWGL (Laurson and Kuuskankare 2002). We utilize a system where high-level score information is converted into control information which in turn is mapped with a visual patch representation of an instrument. Our system allows to build instrument models using a copying scheme for patches. Special synth-plug boxes are used to automatically parametrize control entry points. A musical score is translated into a list of control events. The user can visually associate these events with the instrument definitions by using a mapping scheme. The proposed protocol has many advantages: an instrument definition can easily be modified allowing the user to incrementally improve the model; control entry points and their mapping to score control information can be added or deleted; the compact visual representation is easy to understand and maintain; high-level, editable and musician-oriented score representation allows to generate rich control information that is both precise and repeatable.

This paper presents some of our recent efforts where the aim is to extend the current mapping scheme so that it can cover also real-time applications, where instrument models can be controlled with an appropriate MIDI controller, such as a MIDI guitar or a MIDI keyboard. One of the most important design principles when realizing this extension was to avoid a completely separate system that would be used for real-time applications only. Instead, our starting point was the existing score-based approach where real-time events, such as note-on events, could be translated to a kind of ‘virtual’ score. This information can then be used to control instrument models in a very similar fashion than in the score-based case. Some necessary extensions, such as note-off events and continuous control information, can be added to the visual patch representation without disturbing the score-based control scheme. Probably the main benefit in this kind of approach, where a single patch can represent an instrument definition for both score-based and real-time applications, is that the instrument can be calibrated and refined in a much more flexible and versatile way than before.

We will begin by explaining briefly the score-based patch definition. After this we describe our new formalism, that extends the system to cover real-time situations. The next section gives case studies where some of the existing instrument models are modified so that they can receive control events. Here we give examples how keyboard control information can be mapped to realize different playing styles.

2 Visual Instrument Definitions And Score Control
This section gives a short overview of the main concepts behind the visual patch definition and its relation to score-based control. A more detailed description is given in Laurson, Norilo, and Kuuskankare (2005).

Figure 1 gives a top-level patch definition of a guitar model. A box called ‘copy-synth-patch’ is used to copy a string model—called ‘string’—6 times (the ‘string’ box is an abstraction containing a patch consisting of DSP-
modules that are used to realize the string model). Thus the user has to define the string model only once and the system automatically copies this model as many times as required. In order to distinguish between different duplicated patch instances ‘copy-synth-patch’ generates automatically symbolic references to specific user defined entry points. These entry points are defined by connecting a ‘synth-plug’ box at the leaves of a synthesis patch. The entry-points are used afterwards to control the synthesis process. The symbolic references are pathnames, similar to the ones found in OSC (Wright and Freed 1997), such as ‘guitar1/1freq’ or ‘guitar2/6/freq’.

Thus the user has to define the string model only once and the system automatically copies this model as many times as required. In order to distinguish between different duplicated patch instances ‘copy-synth-patch’ generates automatically symbolic references to specific user defined entry points. These entry points are defined by connecting a ‘synth-plug’ box at the leaves of a synthesis patch. The entry-points are used afterwards to control the synthesis process. The symbolic references are pathnames, similar to the ones found in OSC (Wright and Freed 1997), such as ‘guitar1/1freq’ or ‘guitar2/6/freq’.

Figure 1. A guitar model consisting of 6 strings.

The scheme in Figure 1 is associated to a score by adding to the ‘synth-plug’ boxes information about accessors. These accessors are defined as short methods that are used to access information from a note object, such frequency, amplitude, start-time, and duration. Furthermore, the user gives control labels—‘D’, ‘T’, or ‘C’—to the synth-plug boxes whether they are used for discrete, trigger or continuous control purposes.

From this information (i.e. entry point pathnames, accessors and control labels) the system creates automatically both discrete and continuous control methods for the instrument in question.

3 Real-time Extension

In this section we sketch how the score mapping scheme that was outlined in the previous section is extended to cover real-time situations. When receiving a note-on event the system updates a note object which is sent immediately to the discrete control method of the current instrument. Thus typically real-time control of note-on events work in the same way as in the score-based approach. The note object contains basic information about start-time, amplitude and pitch. This information can be enriched according to the state of the current controller with various expressions, note-heads and articulations. Thus the scheme mimics the score-based approach and the system reacts as if

this real-time information is played from a score. Continuous control can be achieved with a ‘midi-cc’ synthesis box that returns a scaled value of a given continuous MIDI controller. Note-off events, in turn, are realized by adding new labels to synth-plug boxes such as ‘NOFD’, ‘NOFT’ and ‘NOFC’ (discrete, trigger and continuous note-off events). These are translated in a similar way as explained above to control methods which are triggered when the system receives a note-off event.

Figure 2 gives a more complete version of a guitar model that can be used in two modes: (1) MIDI mode (‘midi’) or (2) score mode (‘score’). The current mode can be selected by using a master switch box having the label ‘MasterSW’ in the low-right corner of the box. The master switch box can have one or several slave switch boxes that will follow the state of the master switch. Both the master and slave switch boxes share the same box-string, which is in our case equal to ‘midi/score’.

Figure 2. A guitar model that supports both score-based and real-time control. The current control mode (either ‘midi’ or ‘score’) can be selected using the master switch box at the bottom of the figure.

The Appendix gives the contents of the ‘string’ abstraction box of Figure 2. This patch contains several ‘synth-plug’ boxes that are labeled either with ‘D’, ‘T’, ‘C’ or ‘NOFC’. The latter ‘synth-plug’ box (i.e. with the label ‘NOFC’) is used only when the patch runs in MIDI mode.
Also we see additional slave switch boxes with the box-string ‘midi/score’ (also these boxes follow the state of the master switch box given in the top-level patch of Figure 2).

4 Instrument Model Realizations

Next we give some case studies where we add real-time MIDI keyboard control to some instrument models. While it is obvious that a MIDI keyboard controller is not the best choice for all the discussed instruments we chose it as a test case as it is easily available and does not cause some of the latency problems that are found in other controllers.

The physics based instrument models that we will discuss are the following: a harpsichord model (Välimäki et al. 2004), and a guitar model (Laurson et al. 2001). In both of these cases the focus is somewhat different. In the harpsichord case the implementation is the simplest as we define only a basic keyboard control scheme that reacts to MIDI note-on and note-off events. Controlling the guitar model from a MIDI keyboard is more problematic and we offer several options how this model could be controlled in a reasonable way.

When a real-time instrument is defined it creates a list of note objects. The number of notes depends on the instrument in question. For instance a harpsichord, that has no pedal and where the polyphonic texture is created normally with the fingers only, requires typically 8-10 notes. A standard 6-string guitar requires 6 notes, one for each string.

In order to be able to receive note-on and note-off events an instrument has to specialize at least two main methods: find-next-RT-note and find-current-RT-note-off-note. Both have two arguments: the current instrument and a midi-information list. The former method is called—with the current instrument as the first argument—every time the system receives a note-on event, while the latter method is called when receiving a note-off event.

find-next-RT-note has two main functions. First, it has to find a note from the list of available notes. Second, it has to update the chosen note with some appropriate information. This information can be used later by the discrete and continuous control methods to realize various playing options. Typically the note is updated according to the current midi and velocity values given by the midi-information list.

More interesting cases include writing some expression information in the note that will in turn affect the behavior of the note. This requires that the appropriate accessors for the instrument are able to respond to the expression marking in question.

4.1 Harpsichord

The definition of the find-next-RT-note method for the harpsichord is straightforward. It simply finds the next free note from the list of available notes. If no free notes are available then the note that has the oldest start-time is chosen.

A simple imitation of the lute stop can be simulated by checking the state of the MIDI sustain pedal when receiving a note-on event. If the pedal is pressed we write a pizzicato expression to the note. This change will automatically affect the parameters controlling the overall and frequency dependent decay of the current string model, resulting in a pizzicato-like sound.

4.2 Guitar

The first part of the find-next-RT-note method for the guitar is also straightforward as each note is mapped directly to a string of the model. Thus when a passage of fast notes are played on the same string, the previous notes are always stopped before the next note can begin, even when the player does not lift the fingers from the keys. This results in a more realistic simulation of guitar playing.

In general the guitar model is more demanding from the control point of view than a keyboard instrument. One of the main problems is that there is no obvious way to map keyboard information to a guitar model. Therefore some compromise must be found (obviously the best solution would be to control the guitar model with a low-latency MIDI guitar controller).

One solution is to use a scheme where the channel information of a note-on event is mapped directly to the string number of the model (thus a note-on event on channel 1 would trigger the first string, and so on). In this scheme it is possible to play all potential string/fret combinations of a string and thus this mapping is suitable when calibrating individual strings of the model. The drawback is however that polyphony will be tedious to realize (some keyboards allow the keyboard to have several split-points thus allowing some rudimentary polyphony). Another solution is to map the keyboard so that pitches ranging from E2 to G#2 are assigned to string 6, notes from A2 to C#3 to string 5, notes from D3 to F#3 to string 4, notes from G3 to A#3 to string 3, notes from B3 to D#4 to string 2, and finally all notes from E4 onwards to string 1. In this option several idiomatic guitar chords and textures are playable and the model will sound more ‘guitar-like’ when played from a keyboard. Even a more flexible scheme could analyze the incoming pitches and dynamically calculate a suitable fingerings using a constraint-based approach.

There are several interesting options that can be used to mimic different playing styles. One obvious possibility is to write a pizzicato expression to the current note as was done in the previous subsection. Other options include changing the note-head of the current note to a diamond-shaped note-head resulting in harmonics. Finally, a slur expression would result in a playing style that mimics the left-hand slurring technique. Note that all these note manipulations are already supported by our the score-based system. Thus
these additions do not require any extra programming efforts.

5 Conclusions

This paper presents our latest developments in our synthesis environment that extends the current score-based control scheme so that interaction using MIDI controllers is possible. The visual score-based instrument definitions can be extended to support real-time control by adding switch boxes, continuous MIDI controller boxes, and modules for note-off events. As the real-time system mimics closely the score-based approach sophisticated mapping of raw MIDI data to high-level playing styles is straightforward.

Although the system is functional there are still two main drawbacks in the current implementation: there can only be one real-time instrument active at a time and it is not possible to mix real-time and score-based approaches. Especially the latter problem needs some careful consideration and will be left as a future task.

6 Acknowledgements

The work of Mikael Laurson and Vesa Norilo has been supported by the Academy of Finland (SA 105557) and the work of Henri Penttinen has been supported by the Pythagoras Graduate School of Music and Sound Research.

References


Appendix

A guitar string model where switch boxes with the box-string ‘midi/score’ determine the control-mode of the instrument. In real-time these switches affect the vibrato and note-off properties of the instrument.