Towards Idiomatic and Flexible Score-based Gestural Control with a Scripting Language

Mikael Laurson
CMT
Sibelius Academy
Helsinki, Finland
laurson@siba.fi

Mika Kuuskankare
CMT
Sibelius Academy
Helsinki, Finland
mkuuskan@siba.fi

ABSTRACT

In this paper we present our recent enhancements in score-based control schemes for model-based instruments. A novel scripting syntax is presented that adds auxiliary note information fragments to user specified positions in the score. These mini-textures can successfully mimic several well known playing techniques and gestures - such as ornaments, tremolos and arpeggios - that would otherwise be tedious or even impossible to notate precisely in a traditional way. In this article we will focus on several 'real-life' examples from the existing repertoire from different periods and styles. These detailed examples explain how specific playing styles can be realized using our scripting language.

Keywords

synthesis control, expressive timing, playing styles

1. INTRODUCTION

The simulation of existing acoustical musical instruments such as the classical guitar in this study [4] - provides a good starting point when one wants to evaluate the quality of a synthesis algorithm and a control system. In this paper we aim to present our recent research efforts dealing with our score-based control scheme [8]. Various aspects of our score-based control system have already been presented in different papers, for instance time modification [5], playing technique realizations [9], and the more recent article dealing with macro-notes [6]. In the following we aim to combine these features and show how realistic playing simulations can be realized in an economical way. We will discuss three larger case studies from the existing guitar repertoire and give information how the system is able to reach convincing simulations. The realizations of these examples can be found as MP3 files in our home page: www.siba.fi/pwgl/pwglsynth.html.

Musical scores in our system are situated within a larger environment called PWGL [7]. PWGL is a visual programming language based on Lisp, CLOS and OpenGL. Scores are of primary importance in our system and they can be used in many compositional and analytical applications such as to produce musical material for instrumental music [3].

2. SCORE-BASED SYNTHESIS CONTROL

Our score-base control scheme has several unique features. First, the input process is interactive. After listening to the result the user can modify the score and recalculate the score until satisfied with the outcome. The user can select and edit any range from the score, polish it and hear the refinements in real-time, without re-synthesizing the whole piece. The ability to work with only a small amount of musical material at a time has proven to be very useful. This is especially important when working with musical pieces of considerable length. Second, our system allows to use performance rules that generate timing information and dynamics automatically in a similar fashion than in [1]. The user can, however, also work by hand using the graphical front-end of the notation package. In this case special expression markings can be inserted directly in the score. We have found that this kind of mixed approach - using automated rules and hand-given timing information - is very practical and allows to define time modifications in a more flexible way than using automatic rules only. Third, the system supports both local and global time modifications. The importance of this kind of approach has also been discussed in [2]. Local modifications involve only one note or chord (such as an expression that changes the time interval between notes). A global modification, in turn, handles a group of notes or chords (a typical example of this is a tempo function).

3. MACRO-NOTES

In this section we focus on an important component our control system called macro-note. The macro-note implementation has been revised and it is now compatible with our scripting language syntax. This syntax in turn has been used in demanding analytical and compositional tasks. The scripting syntax has a pattern-matching header that extracts complex score information, thus making it straightforward to produce side-effects in a score.

Macro-notes allow to use notational short-hands which are translated by the control system to short musical textures. In the simplest case this scheme allows to mimic ornaments, such as trills and arpeggios. The reason for introducing the macro-note scheme in our system comes from our previous experiences using musical scores to generate control information. To realize an ornament - say a baroque trill in a dance movement - just by using metrical notation without any abstraction mechanism can be an awkward and frustrating experience. What is worse, the result is typically ruined if the user changes the tempo. Thus, in order to capture the free-flowing accelerandi/ritardandi gestures typically associated with these kinds of ornaments we need better abstraction mechanisms: the system should respond gracefully to tempo changes or to changes in note

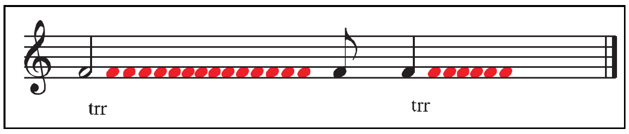

Figure 1: Two macro-note realizations that are labelled with "trr". The auxilliary notes are displayed after the main note as note-heads without stems.

duration; the system should know about the current musical context such as dynamics, harmony, number of notes in a chord; the system should have knowledge about the current instrument and how it should react to various playing techniques.

4. MACRO-NOTE SYNTAX

Next we go over and discuss the main features of the macro-note syntax. As was already stated above, a macro note expression uses our scripting syntax having three main parts: (1) a pattern-matching part (PM-part), (2) a Lisp-code part, and (3) a documentation string. In the following code example we give a simple marco-note script that adds auxiliary notes to the main note simulating a repetition gesture (see also Figure 1):

```
(* ?1  (e ?1 "trr")                    ; (1) PM-part
   (?if (add-macro-note ?1             ; (2) Lisp-code part
             :dur (synth-dur ?1)
             :dtimes '(.13 30* .12)
             :midis (m ?1)
             :indices 1
             :artic 50
             :time-modif
               (mk-bpf '(0 50 100) '(90 130 100))
             :update-function 'prepare-guitar-mn-data))
   "repetition")                      ; (3) Documentation
```

In the PM-part (1) we first state, with a wild-card, '*', and a variable, '?1', that this script is run for each note in the score (thus '?1' will be bound to the current note). Furthermore we check whether the note contains an expression with the label "trr". If this is the case we run the Lisp-code part (2). Here we call the Lisp function 'add-macro-note' that generates a sequence of notes according to its keyword parameters. The arguments are normally numbers, symbols, lists or break-point functions. Internally these arguments are converted to circular lists. In our example we first specify the duration of the sequence (':dur'). Next we give a list of durations (':d-times'). After this we define the 'pitch-field' of our macro-note, ':midis', which is in our case the midi-value of the current note, '(m ?1)' . A closely related argument, ':indices', follows, that specifies how the pitch-field will be read. Here the pitch-field consists of only one pitch and using the index 1 we get a sequence of repetitions. Two time related parameters follow: the first one, ':artic', defines an articulation value (which is in our case 50 percent meaning 'half-staccato'); the second, ':time-modif', is a tempo function, defined as a break-point function, where x-values are relative to the duration of the note (from 0 to 100), and the y-values specify tempo changes as percentage values (100 percent means 'a tempo'). Thus in this gesture we start slower with 80 percent, make an accelerando up to 130 percent, and come back to the 'a tempo' state with 100 percent. Finally, the ':update-function' performs some instrument specific calibration of the generated macro-note sequence. Figure 1 shows two applications of the macro-note script.

5. REALIZATION EXAMPLES

In this section we discuss three case studies. The first one is a tremolo study realization (the original piece was composed by Francisco Tarrega). The result is given in Figure 2. Although this example is now more complex it follows a similar scheme than the previous one. The following script was used to realize this example. Here the PM-part (1) accesses all chords in a score and runs the Lisp-code part (2) if the chord contains the expression with the label 'trmch' (the variable '?1' will be bound to the current chord). The pitch-field consists now of all sorted midi-values that are contained in the chord. The most complex part of the code deals with the generation of a plucking pattern for the tremolo gesture (see the large 'case' expression) This result defines the ':indices' parameter. Here different patterns are used depending on the note value of the chord. For instance, if the note value is a quarter note, 1/4, then the pattern will be '(2 3)', which will be expanded by the 'add-items' function to '(2 1 1 1 3 1 1 1)'. This means that we will use a typical tremolo pluck pattern where we pluck once the second note and then three times the first note in the pitch-field, then the third note and three times the first note, and so on. We use here also an extra keyword called ':len-function' that guarantees that the sequence is finished after the pattern has reached a given length.

A break-point function controls the overall amplitude contour, ':amp', of the resulting gesture. Note that this contour is added on top of the current velocity value.

Finally, we use two parameters that affect the timing of the result. The ':artic' parameter is now a floating point value that is interpreted by our system as an absolute time value in seconds, here 5.0s (by contrast, in the previous example we used integers that in turn were interpreted as percentage values). This controls the crucial overlap effect of the tremolo gesture. 5.0s is used here as a short-hand to say: 'keep all sounds ringing'. The calculation of the final durations is, however, much more complicated (for instance the low bass notes will ring longer than the upper ones), but this will be handled automatically by the update-function. The ':time-modif' parameter is similar to the one in the previous example: we do an accelerando/ritardando gesture during the tremolo event.

```
(* ?1 :chord (e ?1 "trmch")           ; (1) PM-part
   (?if                               ; (2) Lisp-code part
    (when (m ?1 :complete? T)
      (let* ((ms (sort> (m ?1)))
             inds len-function)
        (case (note-value ?1)
          (3/4
           (setq inds (add-items '(4 3 2 3 2 3) 3 1)
                 len-function '(= (mod  len  24) 0)))
          (1/4
           (setq inds (add-items '(2 3) 3 1)
                 len-function '(= (mod  len  8) 0)))
          (1/2
           (setq inds (add-items '(4 3 2 3) 3 1)
                 len-function '(= (mod  len  16) 0))))
        (add-macro-note ?1
              :dur (synth-dur ?1)
              :dtimes '(.13 30* .12)
              :midis (mapcar 'list ms ms)
              :indices inds
              :len-function len-function
             :amp (mk-bpf
                 '(0.0 25.0 25.25 45.0 45.25 65.0 65.25 100.0)
                  (g+ '(40 20 0 30 10 50 20 40) (vel ?1)))
              :artic 5.0
               :time-modif (mk-bpf '(0 50 100) '(90 130 100))
              :update-function 'prepare-guitar-mn-data))))
  "tremolo chords")
```

Our next example is a realization of a arpeggio study by Heitor Villa-Lobos (Figure 3) and the script is quite similar to the previous one. The main difference is that the pitch-field is sorted according to string number and not according to midi-value as was the case in the tremolo study example. The ':indices' parameter is also different: now it is static, reflecting the idea of the piece where the rapid plucking gesture is repeated over and over again.

We combine here two notions of timing control: a global one and a local one. A global tempo function (see the break-point function above the staff that is labelled "/time") makes a slow accelerando gesture lasting for 5 measures. This global timing control is reflected in our script where the local ':dur' parameter gets gradually shorter and shorter.

```
(* ?1 :chord (e ?1 "vlarp")             ; (1) PM-part
   (?if (when (m ?1 :complete? t)       ; (2) Lisp-code part
          (let* ((ms (mapcar #'midi (sort (m ?1 :object T) #'<
                      :key #'(lambda (n)
                             (first (read-key n :fingering)))))))
            (add-macro-note ?1
             :dur (synth-dur ?1)
             :dtimes '(.14 20* .12)
             :midis (mapcar 'list ms ms)
             :indices '(6 4 5 3 4 2 3 1 2 1 3 2 4 3 5 4)
             :artic 1.0
             :amp (mk-bpf
                  '(0.0 25.0 25.25 45.0 45.25 65.0 65.25 100.0)
                   (g+ (vel ?1) '(50 30 10 40 20 60 30 50)))
             :len-function '(= len 32)
             :update-function 'prepare-guitar-mn-data))))
   " Villa-Lobos arp")
```

Our final example, an excerpt from J. S. Bach's Sarabande, is the most complex one, and it is probably also the most delicate one, due to its slow basic tempo. The piece is ornamented with rich improvised textures, such as portamento glides, trills and arpeggios (see Figure 4). In the following we discuss the arpeggio script that is applied three times (see the chords with expressions having the label "carp"). The arpeggio script is similar to the tremolo example as we have a database of plucking patterns. These are organized here, however, according to the number of notes in the pitch-field. Furthermore, the script can choose randomly (using the 'pick-rnd' function) from several alternatives. This results in arpeggio gesture realizations that are not static but can vary each time the score is recalculated, similar to the baroque performance practices where a player is expected to improvise ornaments.

```
(* ?1 :chord (e ?1 "carp")
   (?if (when (m ?1 :complete? t)
          (let* ((ms (sort> (m ?1)))
                 (ind (case (length ms)
                       (6 (pick-rnd
                               '(6 5 4 3 2 1 2 3 4 5)
                                '(1 2 1 3 4 3 5 6 5 6 5 4 3 2 1)
                                '(1 2 3 4 5 6 5 4 3 2 1)))
                       (5 (pick-rnd
                               '(5 4 3 2 1 2 3 4 5)
                               '(1 2 1 3 4 3 5  5 4 3 2 1)
                                '(1 2 3 4 5  4 3 2 1)))
                       (4 (pick-rnd
                               '( 4 3 2 1 2 3 4 )
                               '(1 2 1 3 4 3  4 3 2 1)
                                '(1 2 3 4  4 3 2 1)))
                       (3 (pick-rnd
                             '( 3 2 1 2 3)
                             '(1 2 1 3  3 2 1))))))
            (add-macro-note ?1
             :dur (* 0.95 (synth-dur ?1))
             :dtimes '(.15 30* .13)
             :midis (mapcar 'list ms ms)
             :indices ind
             :artic 5.0
             :amp (mk-bpf
                   '(0.0 0.25 25.0 25.25 45.0 45.25 65.0 65.25 100.0)
                 (g+ (vel ?1) '(50 0 30 10 40 20 60 30 0)))
             :time-modif (mk-bpf '(0 50 100) '(60 150 90))
             :update-function 'prepare-guitar-mn-data))))
   "Bach arp")
```

6. CONCLUSIONS

This paper presents our recent developments dealing with a score-based control system that allows to fill a musical score with ornamental textures such as trills and arpeggios. After presenting the main syntax features we discussed three larger case studies that aim to show how the macro-note scheme can be used in a musical context.

These examples have been subjectively evaluated by the authors (the first author is a professional guitarist), and we consider the macro-note scheme clearly to improve the musical output of our model-based instrument simulations. While this paper concentrates in the simulation of existing musical instruments, it is obvious that our control scheme could potentially be used also to control new virtual instruments.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[1] A. Friberg. Generative rules for music performance: A formal description of a rule system. *Computer Music Journal*, 15(2):56–71, 1991.

[2] H. Honing. From time to time: The representation of timing and tempo. *Computer Music Journal*, 35(3):50–61, 2001.

[3] M. Kuuskankare and M. Laurson. Expressive Notation Package. *Computer Music Journal*, 30(4):67–79, 2006.

[4] M. Laurson, C. Erkut, V. Välimäki, and M. Kuuskankare. Methods for Modeling Realistic Playing in Acoustic Guitar Synthesis. *Computer Music Journal*, 25(3):38–49, Fall 2001.

[5] M. Laurson and M. Kuuskankare. Aspects on Time Modification in Score-based Performance Control. In *Proceedings of SMAC 03*, pages 545–548, Stockholm, Sweden, 2003.

[6] M. Laurson and M. Kuuskankare. Micro Textures with Macro-notes. In *Proceedings of International Computer Music Conference*, pages 717–720, Barcelona, Spain, 2005.

[7] M. Laurson and M. Kuuskankare. Recent Trends in PWGL. In *International Computer Music Conference*, pages 258–261, New Orleans, USA, 2006.

[8] M. Laurson, V. Norilo, and M. Kuuskankare. PWGLSynth: A Visual Synthesis Language for Virtual Instrument Design and Control. *Computer Music Journal*, 29(3):29–41, Fall 2005.

[9] M. Laurson, V. Välimäki, and C. Erkut. Production of Virtual Acoustic Guitar Music. In *AES 22nd International Conference on Virtual, Synthetic and Entertainment Audio*, pages 249–255, Espoo, Finland, 2002.

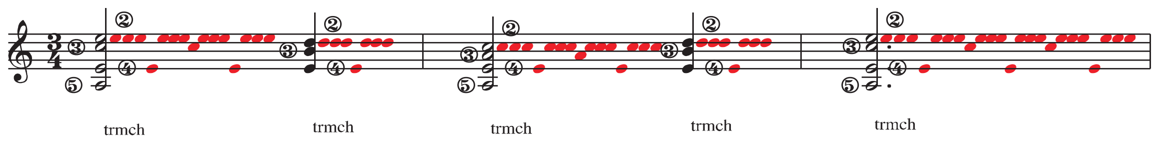

Figure 2: Realization of the opening measures of the tremolo study ”Recuerdos de la Alhambra” by Francisco Tarrega.

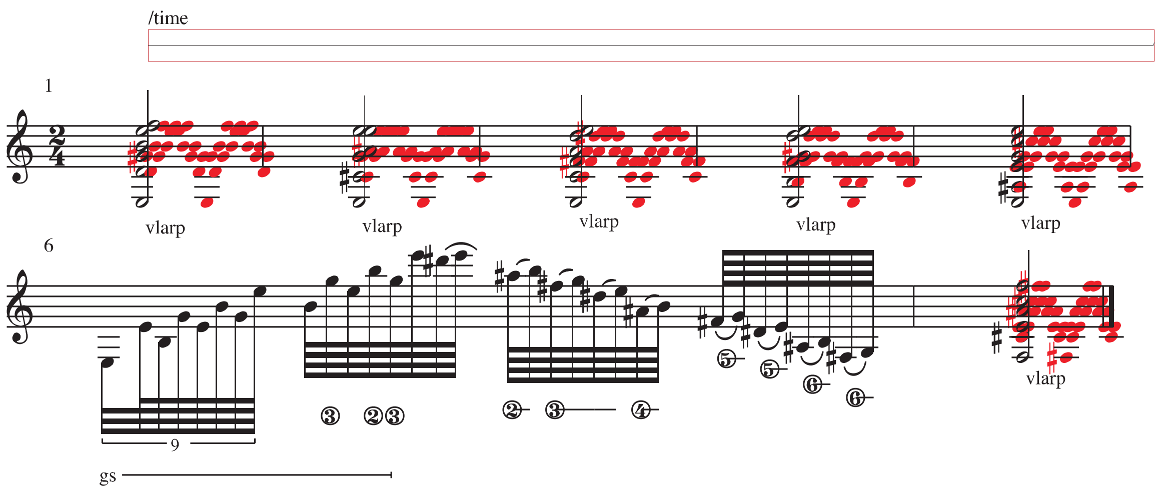

Figure 3: Arpeggio study by Heitor Villa-Lobos. This example is challenging as we use macro-notes mixed with ordinary guitar notation.

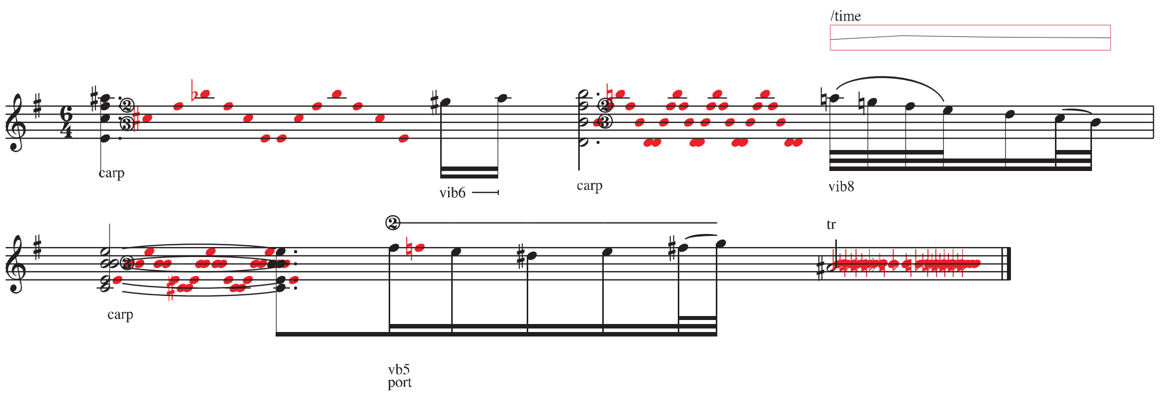

Figure 4: Johann Sebastian Bach: Sarabande. This example contains macro-note arpeggios and trills, vibrato expressions, a tempo function and a portamento expression.