STUDIA UBB MUSICA, LVI, 2, 2011 (p. 43-50)

CHOPIN INTERACTIVE

ADRIAN BORZA[*]

SUMMARY. *How about performing Fr. Chopin's Prelude Op 28 No 7 with an interactive computer?* This is how I began writing a Max/MSP patch for interactive music performance. The computer listens to and performs along with a musician. The patch reacts to different performing approaches of Prelude.

Keywords: Chopin, Prelude, Computer Music, Interactive Music Performance, Max Programming, Live Coding

Listener Objects in Max

Max programming environment supplies several objects that are capable to analyze a live music performance. For example, when a stream of data is sent to a computer from a MIDI keyboard controller, Max objects provide useful information about pitch, loudness, duration, tempo, and many MIDI messages of the ongoing performance.

The Musical Instrument Digital Interface protocol doesn't offer timbre information to produce sounds by its own synthesis instructions. Therefore, the sound generator or the synthesizer is commonly an external hardware, a software synthesizer or VSTi, driven by MIDI.

In this study I will refer to the qualities of sound which were the premise of conceiving my Max/MSP patch[1] for interactive performance of Prelude Op 28 No 7 by Fr. Chopin.

Listen to Pitch and Loudness

A convenient way to extract information about pitch, loudness, and MIDI events, during a live performance, is the sophisticated *borax* object. However, a simpler alternative in this circumstance would be to make use of the basic features of *notein*. This object listens to and then it reports different integers, corresponding to the MIDI *Note On* and *Note Off* messages of the data input stream. It is thus pitch and loudness values.

Using the very same object, an analysis of pitch occurrence can produce valuable data about duration, tempo, and synchronization, among others.

[*] *Currently he is a professor at the Gheorghe Dima Academy of Music. Address: 25, I.C. Bratianu, Cluj-Napoca. E-mail: aborza@gmail.com*

[1] *Chopin Interactive Software* (2010), assembled in Max/MSP/Jitter 5

Definitely, there is musical information that can be transferred from the musician performance to the computer accompaniment. Let's look more closely.

Note Duration and *Note On*

On one hand, duration is the time measured from the beginning of a note (*Note On*) and the end of the same note (*Note Off*), considered as two separate MIDI events. In contrast, tempo is governed by the length of time elapsed from the beginning of a note (*Note On*) and the next note (*Note On*), that is, delta time.

In Max, a *Note Off* is indicated by a *Note On* event with velocity equals 0. To calculate the time elapsed between *Note On* messages with non-zero velocity, I employed the *stripnote* and the *timer* objects to execute the task.

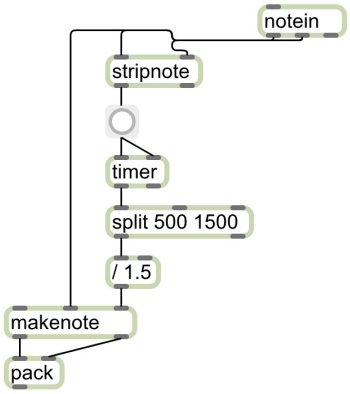
On the other hand, Prelude makes use of just a few duration values, and the accompaniment is quite robust from this point of view. Accordingly, the algorithm for generating quarter notes, interactively, in close relation to the tempo of the performance, takes into account this relative duration value, filtering out other input values with the *split* object.

In the example bellow, note duration (computer) is acquired by manipulating consecutive *Note On* messages (performer). Here's the detailed explanation of the algorithm (Fig. 1):

- *notein* object filters and converts MIDI message into Max *number* message, received from the keyboard controller: the number sent out through the first (left) outlet is the pitch value of the incoming *Note On* or *Note Off* message; the velocity value is sent to the second outlet, 0 for a *Note Off* message.
- *stripnote* object filters out the *Note Off* message received from *notein*, and passes only the *Note On* message to its outlets; the pitch value is transited to the left outlet.
- *button* object sends a *bang* message each time it receives the *Note On* number from *stripnote*.
- *timer* object reports elapsed time, in milliseconds, between consecutive *bang* messages, sent by *button*, in other words, between pitch values with non-zero velocity value, received from the keyboard controller.
- *split* object filters time values sent by *timer;* if the incoming values fall within the range specified in arguments, the object sends those numbers out through the left outlet. In Andante tempo, it means that the output numbers are any duration values ranged from dotted eighth note to dotted quarter note, thus filtering appoggiatura, sixteenth notes, and half notes of the ongoing performance.
- */* (divide) object cuts 1/3 of the values received from *split*, and send them to its outlet, in order to achieve for the computer score, in correlation with *split*, a relatively independent duration values of the notes.

- *makenote* object generates a *Note Off* message after the amount of time specified by */* object; the pitch, which is discussed later, is paired with the velocity value instantaneously received from *notein*.

Fig. 1



Note duration is acquired by manipulating *Note On* messages

Tempo, Synchronization and *Note On*

As long as the *stripnote* object acts like a filter applied to the input data flow, revealing only the pitch values associated with the velocity values ranged from 1 to 127, the task of the smart *follow* object, i.e. to compare the ongoing performance with the recorded performance, is simplified by avoiding any unwanted mismatches. It listens to, it identifies the notes' pitch, it compares both performances in terms of pitch, and then it notifies if the notes are matching each other, as the live performance progresses.

At this point is created an ideal synchronization between the performer score and computer score: in any flexible tempo the performer would play, even with mistakes or gaps, the computer has to follow him, and it does. Every index came from the *follow* object, and passed through the *sel* object, determines the *coll* object to send immediately its recorded information, which represents the pitch values of the accompaniment.

In this example, the solution to tempo and synchronization problems is based on listening and processing the occurrence of *Note On* values. Here is the detailed description of the algorithm (Fig. 2):

- *follow* object searches for the *performace.txt* file's pitch numbers, ignoring other information stored into the file; when the pitch value received from *stripnote* matches the stored value, the index of the matched value is sent out to the left outlet. A sample of *performace.txt* file is presented here:

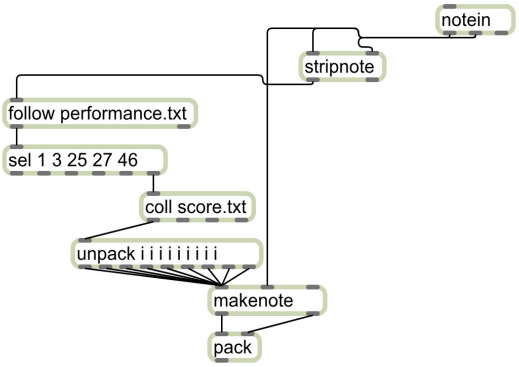
  1382 144 64 60;
  2339 144 73 60;
  3228 144 74 60;

4124 144 71 60;
5038 144 71 60;
6009 144 71 60;
where 64, 73, 74, and 71 are pitch values.

- *sel* (*select*) object selectively passes to its rightmost outlet the index numbers received from *follow*; the object arguments represent single notes, unaccompanied.
- *coll* object sends through its left outlet the information stored at specific addresses into the *score.txt* file; this information embodies just the pitch values of the notes, of the chords. An excerpt from *score.txt* file is illustrated here, according to the format <address, message;>

  1, 0;
  2, 40;
  3, 0;
  4, 52 68 64 62;
  5, 68 64 62 52;
  6, 68 64 52 62;
- *unpack* object breaks up the list of numbers stored after its address, and sends each number to a separate outlet.
- *makenote* object generates a *Note Off* message after the amount of time specified by */* object, as I mentioned earlier; the pitch received from *coll* is paired with the velocity sent by *notein*.

Fig. 2



The solution to tempo and synchronization problems is based on processing the *Note On* values

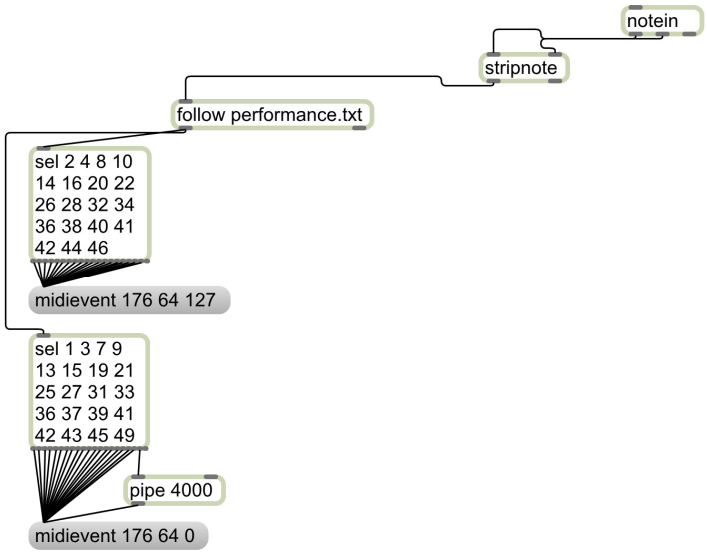
Sustain Pedal and *Note On*

Another step towards achieving the goal of the interactive performance software discussed here is the control of the sustain pedal. A sustain pedal has two states. Because the state of the pedal is either active or inactive, any

value from 64 to 127 is interpreted as active pedal, and from 0 to 63 as inactive. By means of the *message* object, the states can be controlled with particular messages addressed to the *vst~*[2] object: <midievent 176 64 127> to control the active state, respectively <midievent 176 64 0> for inactive pedal.

Each time the *follow* object send an index number, the *sel* objects compare that index with the numbers specified in their arguments, and then they send *bang* messages, one at a time, if the numbers are identical. Consequently, they trigger the messages for activating or deactivating the sustain pedal.

Fig. 3



Activating and deactivating the sustain pedal

Strengths and Weaknesses

Note duration, tempo, note or chord synchronization, and sustain pedal control, associated with the computer score, are the result of computing in real-time the time occurrence of pitch values, identified during performance of Prelude.

There isn't a practical requirement to implement an algorithm for tempo anticipation. Since tempo and synchronization are pitch - dependent, the computer will staidly follow any elastic performance. The advantage is that the performer and the computer are perfectly synchronized. Anyway, the approach it might be disputed, since a human performance reveals the legitimacy of a slightly delay of 5-10 milliseconds or more between the notes of a chord; otherwise they are hardly noticeable by human ear.

[2] The *vst~* object is the host for a VSTi plug-in, a software synthesizer such as Steinberg *The Grand* virtual piano. See also Fig. 4.

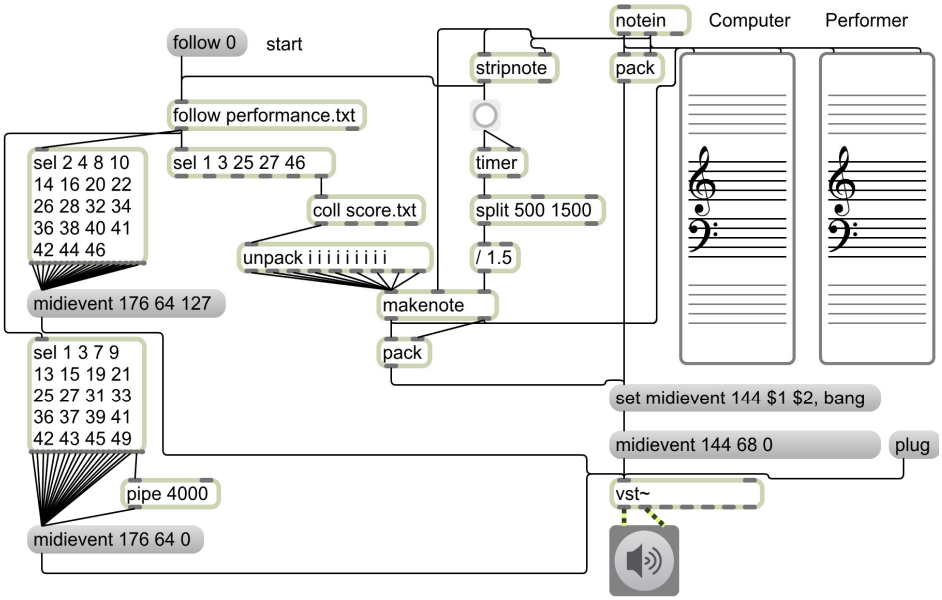
Every note or chord played by computer has the same loudness as the performer's instant loudness. The effectiveness of this solution resides in the fact that there is no need for an additional object, besides a simple connection between two existing objects. The simpler the patch, the speedier it is.

In order to avoid overlapping the same notes, when the computer repeats a chord, their duration is a bit shortened. Thus, the virtual instrument plays all the notes, without exception, back. It is a technological restraint.

The redundant duration values are filtered out from performance; only a relative value of quarter note is transferred to the accompaniment, but it is coupled with the next pitch or chord of the computer score. This naturally happens when note duration is based on delta time. The benefit of this approach is that it produces a relatively independent duration values. Nevertheless, the sustain pedal, as is written into the original score, obliterates the diversity of note duration discrete values. I preferred to make use of it in abundance, in view of the fact that it can be automatically switched on/off, in a fraction of second.

As far as I am concerned, it takes several minutes to assemble the software in Max (Fig. 4). As an example of live coding, the reader of this study can find on the Internet[3] a video recording of the process of writing in real-time this Max patch.

Fig. 4



Max patch for interactive performance of Prelude Op 28 No 7 by Fr. Chopin

[3] Chopin, Prelude Op 28 No 7, Max/MSP (1/2) at the address: http://www.youtube.com/watch?v=iYOps3Gk_WU

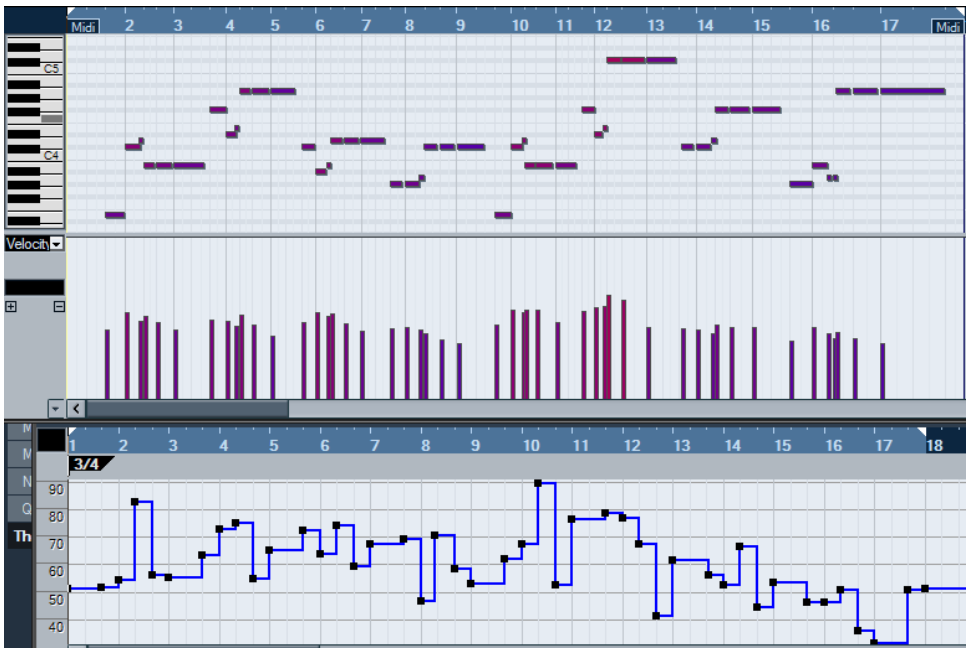
Performer's Score

The musical score performed by musician consists in a single melodic line extracted from Chopin's Prelude. This is the sound outline of Prelude, is the *discantus*. The performer recreates the music by playing this score, listening to computer feedback, imposing his own performance style.

In this respect, I have previously transcribed the recording[4] of Sviatoslav Richter's performance, subsequently emulated in my interactive performance[5] with the computer, which is described further.

As it appears from the Richter's rendition, tempo oscillates very often, modifying each beat of 3/4 bar with values between 31 and 89 (see Fig. 5). The second musical phrase includes both extreme values. Intensity values are distinct, ranging from 43 to 82 of 128 discrete MIDI values. There is a pattern that can be identified: each melodic motif has its particular version of the *crescendo-decrescendo* intensity curve. The culmination is located at the 12th bar.

Fig. 5



Transcription of Sviatoslav Richter's performance: sound outline, intensity, and tempo (downwords)

[4] *Richter the Master, Vol. 10: Chopin & Liszt – Chopin: Prelude for piano No. 7 in A major, Op. 28/7*, 00:00:53, Decca, 1st Jun 1988

[5] Chopin, Prelude Op 28 No 7, Max/MSP (2/2) at the address: http://www.youtube.com/watch?v=ChfJ0GZXUYI

Conclusion

This paper has focused on some of the programming and musical aspects related to the performer score and computer accompaniment, with emphasis in listener objects in Max/MSP. The listener objects are capable of analyzing an ongoing music performance, and of extracting information about pitch, loudness, and diverse MIDI events. A Max/MSP patch is built with the purpose of instantly transferring the information from the musician performance to the computer accompaniment.

The entire process of performing Prelude with an interactive computer was described here as being a precisely set of rules acting on specific incoming data, embodied into a Max/MSP patch. The rules were conceived beforehand, as part of an algorithm, through formalization. The computation is mechanical in nature; the rules mechanically apply to date. The process of performing is thus represented by a series of mathematical and logical operations with numbers.

REFERENCES

***, *Max 5 Help and Documentation*, Cycling ’74, San Francisco.

****, Richter the Master, Vol. 10: Chopin & Liszt*, Decca.

***, *The Complete MIDI 1.0 Detailed Specification*, MIDI Manufactures Association Incorporated, La Habra.

Borza, Adrian, *MIDI Scripts*, Lucian Badian Editions, Ottawa, 2008.

Chopin, Frédérique, *Prelude Op 28 No 7*, Breitkopf & Härtel.

Winkler, Todd, *Composing Interactive Music: Techniques and Ideas Using Max*, The MIT Press, Cambridge, Massachusetts, 1998, 2001.