STUDIA UNIVERSITATIS BABEŞ-BOLYAI, MUSICA, LV, 1, 2010 (p. 49-58)

PROLEGOMENA TO INTERACTIVE MUSIC SYSTEMS

ADRIAN BORZA[1]

SUMMARY. This paper aims to discuss the interactive music system concept. An operational computer with "intelligent" software "understands" the performer actions and "follows" the score, being able to accompany the soloist, to transform the sound, and to generate music, during the ongoing performance. It provides the reader with compositional algorithms for the purpose of illustrating Max/MSP programming methods and techniques.

Keywords: interactive music system, music software programming, algorithmic composition, electronic music

Introduction

As I discussed in *IAC: An Interactive Music System*[2], the features of the Interactive music system are: interactive (depends on the input), works in real-time (reacts instantly to the input), analyses (to data input) and reacts (producing data output), flexible (adapts itself to changing performance situation), algorithmic (uses compositional algorithms), formal system (represents the formalization of musical language).

Message Passing and Data Flow

The flow of time is essential to interactive music, as music is considered a temporal art. The question is: how the composer decodes, from a musical perspective view, the features of the message and data flow, as a result of interpreting the performer actions?

Deciphering the temporal flow consists, among others, in identifying and formulating the compositional problem: to isolate in Max a particular event from a series of musical events during the ongoing performance, with the aim of triggering an action or a process. Most of the time, discerning the matter, clearly and rigorously, should lead to find a suitable solution.

The approach to the compositional problem is algorithmic, without doubt, by a logical inference control, in order to develop the compositional algorithm that is the method or set of rules, which acts on various types of data.

---

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

[2] Borza, Adrian, *IAC: An Interactive Music System*, Studia Universitatis Babeş-Bolyai, Series Musica, LIV, 1, 2009, p. 125

Developing an efficient algorithm it leads to solve the problem, i.e. judiciously selecting internal and external Max objects which are designed to execute individual and precise tasks. Some objects are better than others, in terms of action and communication. Once selected, the objects are graphically interconnected, and then the compositional algorithm is tested and debugged. This is a practical stage of the composition process, and all the composer's programming skills are emphasized. The composer will design and produce countless Max compositional algorithms for a single problem; the ideal would be to find the optimum. If there is an improper understanding of the problem or the composer makes a clumsy decision on selecting and connecting the objects, it may even affect the computer processing speed – vital for interactive music. A general rule in programming is to find simple solutions to complex problems.

Compositional algorithms

"Composer objects represent musical processes, known in computer music as compositional algorithms. […] A canon (round) is a very old compositional algorithm"[3].

The interactive music system has inherent compositional algorithms; however, the graphic user interface and the sound synthesis algorithms will fulfil it. A comprehensive debate of compositional algorithms is beyond the purpose of this document, while I have tried to give an introduction to interactive computer music systems. The algorithms embodied into this study could be useful to those unfamiliar with Max programming.

The following compositional algorithms are designed to react in real-time, interactive to unpredictable events, such as music improvisation and spontaneous composition. Another feature of the compositional algorithms is mapping; the performer action is associated to the computer process, and one musical parameter transforms another parameter. For example, the pitch is captured and it changes the duration, the intensity is intercepted and it modifies the delta time[4], and so on. In this regard, see *B. Mapping algorithm* that increases and decreases the speed of pre-recorded sequence on computer, by means of the musical interval size, performed by soloist.

I have chosen to discuss in detail tree basic compositional algorithms implemented in Max, which speak about **unpredictable temporal message and data flow**. The following examples illustrate my personal programming style and experience.

---

[3] Winkler, Todd, *Composing Interactive Music: Techniques and Ideas Using Max*, The MIT Press, Cambridge, Massachusetts, 1998, p. 173

[4] The number of milliseconds elapsed since the previous *Note On* event (Zicarelli, David, *Max User's Manual – Reference Manual*, Version 4.6, 2006, p. 77)

Practical problems

When identifying into a MIDI data stream

- *any musical event, in the low range,*
- *a melodic ascending interval, between C 3 and C 4,*
- *a minor chord, in mp,*

It must trigger

- *the playback of the pre-recorded musical sequence*
- *the rhythmic augmentation/ diminution of the pre-recorded musical sequence*
- *the melodic synchronization, event by event, of the pre-recorded musical sequence (tempo, duration, and intensity synchronization)*

A. Playback Control

1. Any musical event, in the low range,
2. Triggers the playback of the pre-recorded musical sequence

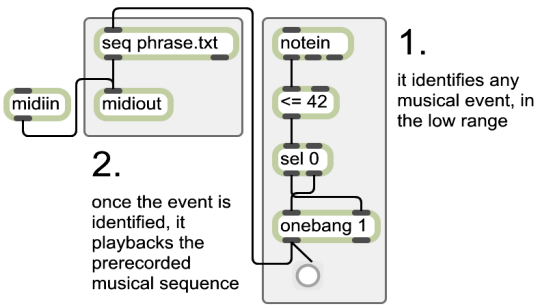
The *Playback Control* compositional algorithm (Figure 3) has a high degree of generality; it endeavours to solve a wide range of synchronization issues between performer and computer. This refers to the control of the playback of any pre-recorded MIDI sequence, by the use of a musical event – single note, melodic or harmonic interval, melodic structure, chord, cluster etc. – performed in the low range of the MIDI instrument. The pre-recorded sequence could be a musical micro-structure, a musical phrase, a series of chords, or may have any structure, extension and complexity.

There is one significant analysis sub-algorithm operation (Figure 3, 1.), since it produces the expected response at the time of the first sound of a block of sounds emerges, avoiding in this way, unexpected interruption during the playback of the pre-recorded sequence (see below for a detailed description of the sub-algorithm).

By inserting additional commends and objects, the compositional algorithm may be developed so as to automate the playback of a complete series of pre-recorded sequences.

Fig. 3

Playback Control



The analysis sub-algorithm operations are executed step by step, from top to bottom and from right to left, in a critical order of a reliable functioning:

1.1. The *notein* object identifies and selects only the *Note On* and *Note Off* messages which constitute the temporal flow of input data. The object converts and transmits *int* messages that are integers corresponding to the pitch of the sound.

1.2. The values of the *int* messages received by the <= object are compared as they arrive with the number of his argument, 42. This value represents the upper limit of the low sound rage, expanded from 0 (C-2) to 42 (F # 1); C3 in MIDI system is central C. If it meets the condition, i.e. numbers are less than or equal to 42, the object generates one *int* message with value of 1, for each low sound, but if the statement is false, the generated *int* message has the value of 0.

1.3. The *sel* object compares the values of 1 and 0 with its argument, 0. If the tested *int* message is 0, then a *bang* message is sent through the left outlet, otherwise, if the number of the *int* message is 1, then this *int* message, rejected after testing, is sent through the right outlet, pointing to low pitches.

1.4. The two cables are cross coupled with the *onebang* object, with the argument 1, so that it allows passing a single *int* message, which subsequently converted, will trigger the pre-recorded music sequence. More precisely, the *bang* message, received by the right outlet, designating that the pitches are not low, along with argument 1 of the *onebang* object, initializes the object as if it has already received a *bang* message through the right outlet, allowing a single pass of *int* message, others are stopped. Consequently, it is converted to a *bang* message, and is sent to the *seq* object (Figure 3, 2.), which triggers the playback of the “phrase.txt” file. Thus it is possible to isolate to the first sound of a musical event, or of a package of sounds; middle and high pitches were rejected by the action of object <=, described above in 1.2.

B. Mapping

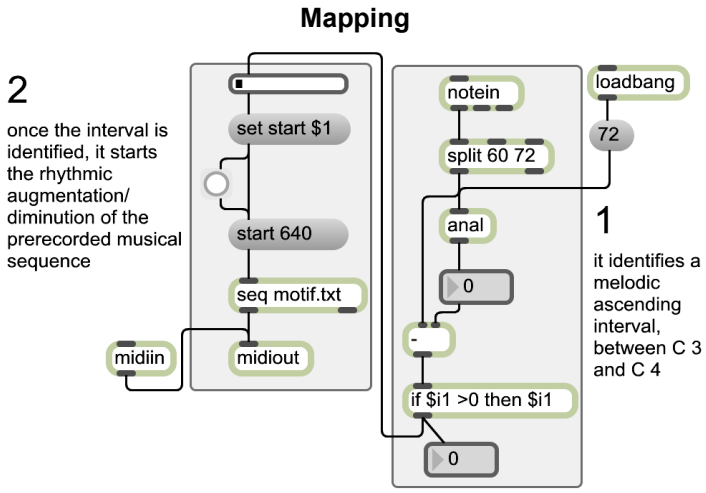
1. A melodic ascending interval, between C 3 and C 4,
2. Triggers the rhythmic augmentation/ diminution of the pre-recorded musical sequence

The *Mapping* compositional algorithm (Figure 4) is more selective and is addressed to a particular synchronization situations and performer/ computer interaction. The algorithm brings in the solution by identifying, from the multitude of musical events produced by the instrument, an ascending melodic interval

located in the first octave, between C 3 and C 4 in MIDI system, on one hand, and it calculates the interval size, on the other hand. The aim is to control the playback of the pre-recorded musical sequence, and also to modify it by means of the rhythmic augmentation and diminution, proportionally to the identified size interval. The sequence can be anything in an MIDI format. In addition, the algorithm selects a harmonic interval, respectively the upper interval of the chord structure of three or more sounds.

For developing purposes of the algorithm, it may be redefined the selection type of the captured musical event in a restrictive way, to a melodic interval, thus being ignored the chords and the harmonic intervals. An easier solution is to automate the on/ off function of the sub-algorithm, using *toggle* and *gate* objects, for example. Another approach, more advanced, is to filter the events perceived as instantaneous, using the information produced by the *thresh* object.

Fig. 4



The calculus operations of the analysis algorithm (Figure 4, 1.):

1.1. The *notein* object allows transiting only the *Note On* and *Note Off* messages which are sent by the MIDI instrument through the interface. These messages are converted in *int*, and the numbers represent the pitch.

1.2. The *split* object is looking and forwarding the numbers located within the limits specified by its arguments, with values of 60 (C3) and 72 (C4). As a result, the object ignores the *int* messages with values smaller and equal to 59, likewise greater and equal to 73, therefore the pitch who is not in part of the first octave.

1.3. In order to analyze two consecutive numbers, *anal* object saves and transfer integers to the left, from state 2 to state 1, in a *list* message, which is generated and sent to the outlet. This transfer is important for the arithmetic calculation performed in Section 1.5.
1.4. The *number* object displays and transmits only the first numerical value, by filtering and converting the *list* message in *int* message.
1.5. The - object (subtraction operator) performs the arithmetic computation of two *int* messages. By the right inlet, it is inserted the first operand (prior), received from the *number* object, without causing the computation. The operand represents the *Note On* message. The second operand (rear), which is a *Note Off* message, actually runs the subtraction computation and sends the result, once the message is received from the *split* object, by the left outlet.

Example

The message and data flow when *Note On* (60/C3) is transmitted:
>> *int* (60) from *notein* to *split*…
>> *int* (60) from *split* to *anal*…
>> *list* (3 arguments) from *anal* to *number*…
>> *int* (72) from *number* to -…
// comment: C4 is transmitted to the prior operand (right outlet) of „subtraction" object //
>> *int* (60) from *split* to -…
// comment: C3 is transmitted to the rear operand (left outlet) of „subtraction" object //
>> *int* (-12) from - to *if*…

The message and data flow when *Note Off* (60/C3 with velocity of 0) is transmitted:

>> *int* (60) from *notein* to *split*…
>> *int* (60) from *split* to *anal*…
>> *list* (3 arguments) from *anal* to *number*…
>> *int* (60) from *number* to -…
// comment: C3 to prior operand //
>> *int* (60) from *split* to -…
// comment: C3 to rear operand //
>> *int* (0) from - to *if*…

1.6. The *if* object evaluates the expression it holds. The expression contains a conditional statement: if the *int* message is greater than 0, then it sends this value. If the condition is true, the positive numbers are validated, representing ascending intervals, otherwise, values are

ignored. Validated size intervals from 1 to 12, are used to determine the augmentation and diminution proportion of the processing sub-algorithm (Figure 4, 2.). As the interval value is higher, the duration is less, and the rhythm gets faster; reciprocally, as the interval value is lower, the duration is greater, and the rhythm gets slower.

C. Automated accompaniment

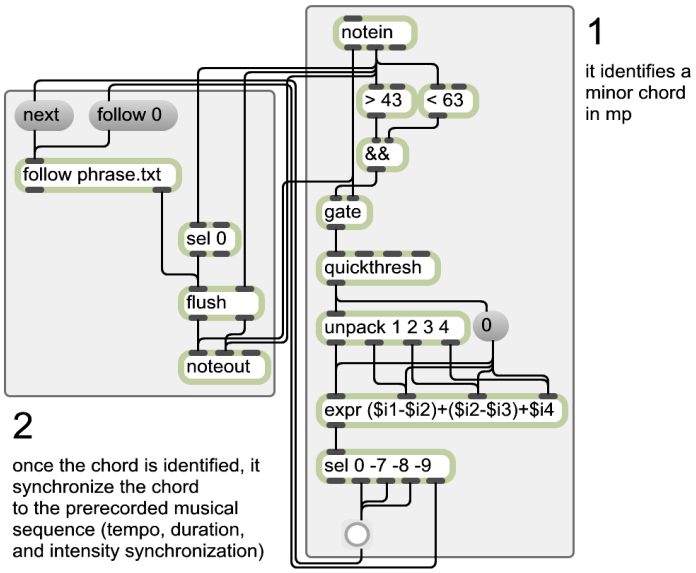
1. A minor chord, in mp,
2. Triggers the melodic synchronization, event by event, of the pre-recorded musical sequence (tempo, duration, and intensity synchronization)

This example of automated accompaniment (Figure 5) could join the compositional algorithms family, with a long history in fact, which fulfil one of the composer's dreams: the computer, equipped with "intelligent" software, "understands" the performer actions, and "follows" the performers' score, note by note, in sync with the computer's score.

The *Automated accompaniment* compositional algorithm aims to synchronize, in terms of tempo, duration and intensity, a random sequence of minor chords in mp performed on the MIDI instrument, with a pre-recorded melody played back by the computer. The main purpose of this algorithm remains the analysis and configuration of the message and data temporal flow received from the performer.

Fig. 5

Automated accompaniment



The calculus operations of the analysis of the algorithm (Figure 5, 1.):

1.1. The *notein* object filters the *Note On* and *Note Off* messages, converts them into *int* messages, and sends them to the middle outlet, then to the left outlet; numbers represent intensity and pitch values.
1.2. – 1.3. The < and > objects (relational operators) compare the values of their arguments, 63 and 43. If the declarations are true, both objects transmit from right to left one *int* message with value of 1. If the declarations are false, the message sent has the 0 value.
1.4. The numbers are compared by the *&&* object (logical operator). If both input values are equal to 1, then the message *int* with value of 1 is sent out, which means that the condition is confirmed. The number 1 indicates also that the intensity has values between 43 and 63 – the relative values of mp. Otherwise, if one of the received numbers is 0, the condition is false, then the *&&* object sends out a 0.
1.5. The *gate* object acts as a "traffic controller". On receiving the 1 value in the left outlet from *&&* object, *gate* object allows the transmission of *int* messages entering the right outlet, received from the *notein* object, that is transferring only the sounds in mp. The *gate* object stops the other pitch values, if 0 is received in the left outlet from *&&* object.
1.6. The *quick thresh* object is optimized to detect chords. In music theory, one of the chord characteristics is simultaneity. But in the musical practice, there are often delays countable in milliseconds between the occurrences of sounds that make up the chord. In Max, these delays between two consecutive *Note On* messages are expressed in delta time values. The *quick thresh* object collects and sends the pitch values validated by *gate* object, in the form of *list* message, if they occur within a period of up to 40 milliseconds, calculated as time delta.
1.7. The *message box* object transmits 0 each time a pitch is identified.
1.8. The values of the *list* message are unpacked and sent after conversion, one by one, as *int* message, through the four outlets of the *unpack* object.
1.9. The *expr* object evaluates an expression alike C language. It calculates algebraic sum of variables or random intervals that compound the chord, which may be of three or more sounds: (x-y)+(y-z)+0.
1.10. The *sel* objects selects the result, i.e. *int* messages with values of -7, -8 and -9, which are the minor triad-chords, in root position, second inversion, respectively in first inversion, and then transmits, for each value, a *bang* message. Every message helps to playback the pre-recorded sequence stored in the "phrase.txt" file, by the specialized *follow* object (Figure 5, 2.). Other results different from 0 are submitted by the right outlet.

How the Performer's Chords and Computer's Melody Are Synchronized?

Note that there is a subordination relationship of the interactivity –the computer follows the artist – despite the fact that, at first sight, we would be tempted to attribute the melody only to performer.

Back to the synchronization mechanism, as mentioned in the paragraph 1.10., the minor triad-chord, in any state would be, in addition, any register would be, but performed in close distribution, is triggering to send a *bang* message, at the end of completing the operations of the sub-algorithm.

After the *follow* object is automatically switched in "follow" mode, using the *follow 0* message, in conjunction with the rejection of an event which is different from the expected minor chord, the *bang* message of the *button* object, coupled with the *next* command sent it to the *follow* object, makes the playback of *Note on* messages of the "phrase.txt" file to advance one step. At each a new *bang* message is sent, therefore, at each a new chord is confirmed by the analysis sub-algorithm, the playback will advance by one step to the next *Note On* message. From musical point of view, the message passing and data flow are described as follows: the artist creates and performs rhythmic formulas and chords, which drive a distinct tempo and rhythm to the pre-recorded sequence. In Max programming, advancing from *Note On* message to the next *Note On* is possible due to the *follow* object instruction to omit all MIDI messages except *Note On*. The chord duration performed on the instrument is transferred to the sequence played by computer, using *flash* and *sel* objects, thanks to the *Note Off* messages sent as *int* messages by the *note in* object. The intensity values of the *note in* object are combined with the *Note On* values of the stored sequence, within the *flash* object, and then are transmitted to the *note out* object. All of these operations lead to synchronize, interactively and in real-time, in terms of tempo, duration and intensity, the performer's chords, created or improvised by the musician, with the computer's melody.

Conclusion

The interdisciplinary-specific Music and Technology professions are considered worldwide to have a bright future. Over 110 universities and music research centres from almost 30 countries have included in their academic programs the interactive systems and the Max programming courses, with the aim of diversifying their educational offer, focusing on the recent musicians' needs.

In Romania, the introduction of such an innovative course would mean:

- ✓ attracting more students into existing specializations,
- ✓ increasing the competitiveness of graduates on the labour market,

- ✓ global educational integration (upgrading),
- ✓ and especially an opportunity for Romanian students to become familiar with Max programming methods and practices.

Below I provide the reader with information regarding places, institutions/ departments and the names of the courses offered in August 2009, concerning Max/MSP and interactive music systems: **Paris**, Institut de recherche et coordination acoustique/musique – *Interaction temps réel*; **Londra**, Middlesex University, **Lansdown** Centre for Electronic Arts – *Interactive and Algorithmic Systems*; **Zürich**, School of Music, Drama and Dance – *Live Electronics and Interactive Composition with Max/MSP*; **Irvine**, University of California, Department of Music – *Interactive Arts Programming*; **Montreal**, McGill University, Music Technology Department – *Interactive Music Systems* (MUMT 610); **Zagreb**, Center for Algorithmic Music – *Composition and Multimedia in MAX environment*; **Seul**, Hanyang University, School of Music – *MIDI and Real Time Programming with Max and MSP*.

REFERENCES

Borza, Adrian, *Muzică şi calculator (Music and Computer)*, Editura Muzicală, Bucharest, 2008

Rowe, Robert, *Interactive Music Systems: Machine Listening and Composing*, The MIT Press, Cambridge, Massachusetts, 1993

Rowe, Robert, The Aesthetics of Interactive Music Systems, in *Contemporary Music Review*, 1999, Vol. 18, Part 3

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

Zicarelli, David, *Max User's Manual – Reference Manual*, Version 4.6, 2006