Macs/s - Modular audio composition syntax/system

Søren Jakobsen
Software engineer, independent researcher
Copenhagen, Denmark
skj@jakobsen-it.dk

ABSTRACT

This article describes the survey, reasonings and outcome of a design project in relation to an idea of a certain method of composing and performing electronic music. Macs/s is a tool to support that method and comprises a simple musical notation language (syntax) for describing pieces of electronic music, and a program (system) that generates audible, interactive music as described. One central design goal of Macs/s is to facilitate 'modular' descriptions, i.e. descriptions formed of several components (modules) that may be referenced and interchanged by other components of the same type - possibly in a live performance situation. A preliminary usability study [hopefully] seems to indicate that Macs/s may help some composers develop their art into new and pleasing directions.

1. INTRODUCTION

As described by Peter Naur [8] the way we humans think about our problems and tasks at hand is tightly connected with the tools and methods we know. A problem is so to say often 'developed' in unison with the tool to solve that problem or task. This is especially clear in the case of creating (electronic) music where artistic practices often involve developing one's own tools and methods. And historically several genres of music have emerged along with the invention of new types of instruments, e.g. with the invention of the electric guitar and the TB303/TR808. As a means to support the evolution of electronic music it is thus interesting to try to invent new types of tools and methods which have perhaps not been thought of or implemented in the same way before. However, as suggested by Ge Wang's NIME 2017 keynote 'The ME in NIME' [11], the design of such tools should also be connected with some ideas about music and 'musical expression'.

2. SURVEY

The design of Macs/s - and the following descriptions - is based on the following analysis and survey results about the work processes behind creating (electronic) music, and about some challenges it can involve.

Some ways of creating music may on a very general level be thought of as involving the following activities.

1. **Sound design**. For traditional (acoustic) music 'sound design' can involve selecting the types of instruments with which the music is to be performed. Electronically produced sounds may also be shaped in vastly varied ways.
2. **Limiting form and genre**. As a form of communication most music to some degree adheres to certain culturally established norms about form and genre.
3. **Methodical development**. A composer may more or less consciously develop own preferred methods, techniques and rules for composing music.
4. **Writing motifs**. Many methods of composition - especially classical, western music - involve writing motifs/themes which constitute a conceptual core of a composition.
5. **Reworking motifs**. A composition may be expanded in time by continuously repeating and combining different variations (modulations) of the core motifs, establishing coherence and 'meaning' in the music. The music of J.S. Bach contains many examples of this.
6. **Performance**. Whereas all the above acts of composing can be seen as ways of limiting the possibilities of a musical performance, the performance is where a human and/or a machine - given more or less freedom - create the actual, audible expression.

In relation to making electronic music the following challenges may be observed.

1. The composer's 'problem' of applying limitations within a scope of possible musical expressions may - as discussed in Robert Henke's CIRMMT lecture 'Give me limits! Two perspectives on...' [3] - be especially challenging in the case of electronic music. Not only do modern electronics give the composer ever expanding possibilities - the composer is also given control over activities which have traditionally been divided between people being experts in each of their fields. The challenge is thus not only to try to master all these activities, but also to maintain focus on each and all of these activities. As expressed by composer T.J. Hertz to The Quietus [1]: "*I used to have this sign on my wall, right behind my monitor, that I'd scrawled to myself in a fit of frustration, that said 'Stop fucking around with the kickdrum [designing its sound] and make some music'.*".
2. General-purpose audio/music programming systems such as ChucK, Csound, SuperCollider, Max and Pure Data provide great possibilities for methodical development, i.e. for developing tools that facilitate and limit one's focus on specific methods of composing

music. Some technically capable composers of electronic music may take advantage of this, but focus in such cases is not likely to be on providing usability and documentation as for sharing publically [7]. Some original methodic developments with tool support that have become publically available (and very popular) include Godfried Toussaint's Euclidian algorithm [10] and Robert Henke's method of 'pattern switching', which is now supported in Ableton Live [2]. There however remains a technical hurdle for many composers wishing to develop new composition methods by these means.

3. A multitude of widely used music software with 'graphical' user interfaces (controlled more via the computer mouse or touchscreen than via the keyboard) such as Ableton Live, Logic, Cubase, Reason, Renoise, etc. may offer better usability for some tasks related to composing music. Even more so perhaps when dedicated controller hardware is used, e.g. as with Elektron Octatrack and Maschine. Some composers of electronic music however seem to find that such tools do not support their creative process well. Composer Tom Jenkinson describes his process of composing with an emphasis on the act of imagining and arranging the music 'in his head' [9]: "*[software-based sequencers] make my brain shut down. When the graphical information is too vivid, it makes it harder to retain the information in my memory, and one critical thing about making music is to have a real-time virtual image of the studio operating in your head, so you can make your choices very quickly*".

4. Humans may play a lesser role in the performance of electronic music, making the performance less pleasing for other humans to experience. Giving a human performer more freedom to interact with and control the composition might potentially improve the performance. However, one could argue that a truly captivating performance involves entering a special 'performance state of mind'. Jan Klug in his research blog [4] describes his own experience of performing, entering a certain state: "*Auditory perception is increased dramatically; incoming audio information is treated differently than when in other states. Less analyzing and interpreting takes place; [...] Visual perception decreases dramatically; the eyes are either fully or half closed [...] conscious interpretation of visual information is minimized and reserved for certain controlling tasks. [...] Conscious attention however is given only in a training situation [...] in the moment that creative thought appears, visual perception is diminished.*"

3. VISION

From the above considerations - and also a bit inspired by the work of Alex McLean [5] - has emerged a vision of a tool for composing and performing music that

1. is more limited in capability and less agnostic in terms of composing methods than commonly used music software.
2. provides high usability for the tasks of a specific method of composition.
3. is flexible enough as to not be considered a musical composition of itself, as e.g. Anders Monrad's Sounding Images [6].
4. supports a method of composing which utilizes the general programming concept of modularity as a means for reworking musical motifs.
5. supports text editing software as a means to support a creative writing process where different ideas may quickly be sketched, edited, compared, arranged, etc.
6. lets users focus solely on non sound design related activities.
7. facilitates a way of creating music where reworking motifs is to a higher degree part of the performance.
8. facilitates a way of performing which requires less attention on the tool itself.
9. may not have been thought of or implemented in the same way before (see Section 9).

4. METHOD

In unison with the development of Macs/s - a process of numerous iterations - a method of composing and performing has been developed. The method relies on an idea of a piece of electronic music as a combination of at least one of each of the following types of interchangeable components (modules).

1. Collection of sounds ('samples', audio fragments) where each sound may have a number of timbral variations.
2. Base rhythm - a time cycle defined by a number of periodic pulses (beats), a tempo/frequency (BPM) with which the pulses recur, and - per sound - a number of divisions per cycle.
3. Abstract sequence - a sequence of numbers that may reference steps in scales of pitch, volume, time offset, etc.
4. Parametric sequence connecting specific sound parameters - pitch, volume, time offset, etc. - with abstract sequences by which the each parameter is to be 'animated'. Each step of a sequence corresponds to a division of a cycle.
5. Sounding sequence connecting specific sounds to specific parametric sequences.
6. Abstract scale - a sequence of numbers which for each step may describe a sound (sample) playback rate, an amplitude, a time offsets, etc.
7. Parametric scale connecting specific sound parameters with abstract scales describing how each parameter is to be configured per step of the scales.
8. Sounding scale connecting specific sounds to specific parametric scales.

Figure 1 shows with arrows how the definitions of some components may reference other components. The alphanumeric codes refer to how components of each type is named and referenced (see Section 5).

The method by which the components - except the sound design - are described involves using standard text editing software to write text that adheres to the rules af a certain syntax (see Section 5). This work may be considered part of the compositional phase of creating music. The performance - where selected components are used to create audible music - involves a human selecting - by entering letters with the computer's keyboard - different combinations

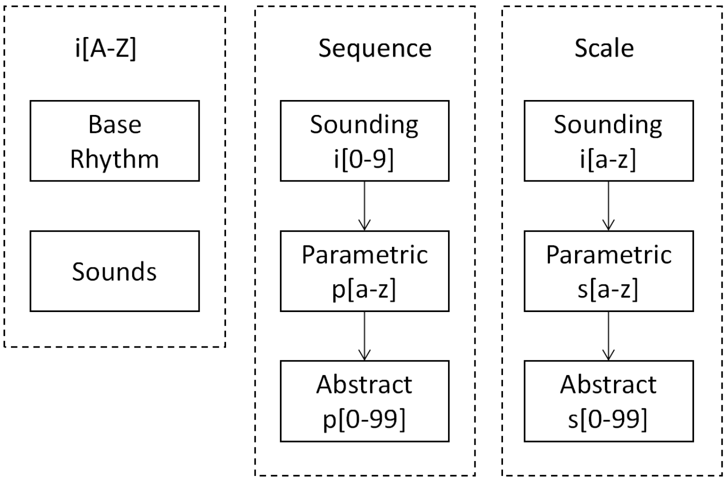

Figure 1: Components of a composition

of 1) a combination of sounds and a base rhythm (capital letters A to Z), 2) a sounding sequence (numbers 0-9), and 3) a sounding scale (letters a to z). The Macs/s system then has enough information to generate audible music.

The creation of music is as such divided into three phases and work contexts:

1. Sound design by use af any audio editing software.
2. Composition by using the Macs/s syntax and any text editing software.
3. Performance by using the Macs/s audio generation system and its performance interface.

The separate work contexts may help the composer stay focused on non sound design aspects of composition when that is the intention. The more limited possibilities of the performance interface - and that it requires less human attention and 'analytical thinking' - may help the performer to enter a 'performance state of mind'.

The concept of modular scales has several possible uses:

1. Defining unusual, e.g. micro-tonal scales.
2. 'Mixing' sounds, e.g. applying scales where the range of volumes is higher or lower.
3. Reworking motifs, i.e. modulating a basic sequence in different ways by applying different scaling modules, e.g. 'mirroring' a tonal sequence by applying a mirrored scale.

An important feature of many types of performed music is the use of gradual changes, e.g. by volume (crescendos) or electronic frequency filtering. The method of defining sequences separate of a 'base rhythm' allows for transitions between sequence components by gradually changing the weighing of an average between the values of two sequence components - and likewise for scale components. The user interface for inputting the weights could be two continuous controllers - sliders og 'turn knobs' - or one 'x/y pad'.

Whether the computer keyboard in combination with text editing software is more suitable for supporting creative work processes than a more 'graphical' interface, is a questions which could be further examined. A reason to believe it could be the case is the immense popularity of text editing software as a tool for working with any kind of text, whereas 'graphical' music software is not an obvious choice for many musicians. The complexity of existing general purpose audio programming systems may have kept many artists from using text-based interfaces, but for those artists Macs/s could be a solution to achieve better support of their creative processes through text editing software.

In relation to the discussed challenges of creating electronic music one may notice that the method does not help composers overcome the technical hurdle of developing (tool support for) personal methods of composing electronic music. Macs/s however does give composers one more (public) tool and method to choose from.

5. SYNTAX

In the following a textual syntax to support the above method is described. Support of this syntax is to be considered the defining feature of a Macs/s implementation.

With the syntax each component/module of a composition may be described by one line of text, where the first one to three characters specify the type of module and gives it a name by which it may be referenced, e.g. a sounding sequence may be defined by a line of text which starts with 'ia', where 'a' will be the name of the module, and a base rhythm by 'iA', where 'A' will be the name. Codes for each type of module is given by Figure 1.

A sound module is defined by creating a file system directory with a number of samples, where the first character of the file name specifies a name of the sound and the next character specifies the index of different variations of that sound. Sounds must be named consecutively by characters a to z, and variations by numbers 0 to 9.

Following is a minimal example of the textual part of a composition where only one module of each possible type is defined. The text describes a cyclic base rhythm of four pulses and a tempo of 100 pulses per minute, using a sound module named 'soundmodule1' - where the first sound will be sequenced to a 'grid' of 16 divisions of the cycle. The sounding sequence connects the first sound to the parametric sequence 'pa', which connects five different parameters to the same abstract sequence 'p0' (scale step 5 will be used for all 16 divisions). 'iq' connects the first sound to 'sa' which connects all parameters to 's0'. Abstract sequences refer to a nine step scale by values 1-9 and abstract scales define values in the range [0;1] - '0.' is implied in the values, i.e. 125 means '0.125', and '.' means '1'. Table 1 shows how values for each parameter type is interpreted.

Table 1: Parameter codes and interpretation af values in range [0;1]

Code	Parameter	Value interpretation
r	Sample playback rate	Half to double rate (octave below and above).
s	Sample index	Mapped to the range of sound variations defined (rounding to nearest).
v	Volume	Silent to full, values >= 0.1 restarts playback of the sample.
o	Timing offset	No offset to offset the length of one pulse.
p	Panning	From left to right speaker (stereo setup assumed).

```
iQ 100 4 soundmodule1 16
i1 a
pa r 0 s 0 v 0 o 0 p 0
p0 5
iq a
sa r 0 s 0 v 0 o 0 p 0
s0 0 125 25 375 5 625 75 875 .
```

Note that the system defines several of the above components by default, so the composition could actually have been written in one line, 'iQ 100 4 soundmodule1 16'. Also note that the performance system by default selects iQ, i1 and iq, which is why they must be defined.

The following example illustrates some more features of the syntax. Note the following.

1. iW: '.' (dot) may be used to specify that a sound should not be playing.
2. i1: two or more parametric sequences (cb...) may be combined, and values of the latter sequence will override the previously given values.
3. pa: not all parameters need to be specified, by default 'x 0' - where x is a parameter - will be defined.
4. p2: several parts, divided by line breaks and indentation, may be specified. Each time the cycle is repeated the part playing is changed.
5. 'Comments', i.e. information that is ignored by the audio generation system, may be started with '-' and ends where the line ends.
6. pb: for each parameter a second abstract sequence may be specified. The first sequence is of 'static' values, i.e. values that do not change over the course of playback of a sample. The second sequence is of 'dynamic' values, i.e. values that may change - with a fixed, quick change rate - while the sample is playing.
7. syntax highlighting will improve readability of the composition.

```
iQ 100 3 soundmodule1 11 7
iW 100 4 soundmodule1 . 14
- patterns
i1 a cb
i2 b ca
pa r 1
pb r 2 3
pc o 3
p1 687
p2 132
   3838
p3 1414
- scales
iq a a
iw b b
sa o 2        - r 0 by default
sb o 2 r 1
s1 . 875 75 625 5 375 25 125 0    - s0 mirrored
s2 0 01 02 03 04 05 06 07 08
```

6. IMPLEMENTATION

A current implementation of Macs/s is using a client/server architecture. The client is implemented as a package for Atom (the text editing software) and the server as a Csound (.csd) file. The client provides syntax highlighting and compilation of Macs/s files, and the server generates audible, interactive music from the compiled files. The client/server architecture allows for a method of working where parts of a playing composition may be updated without interrupting an ongoing musical flow (aka 'live coding').

Implementing the client as an Atom package has been motivated by the following.

1. Support on multiple platforms, OS X, Windows, Linux.
2. Integration with open source development platform GitHub.
3. Use of a well-known language (JavaScript).
4. Syntax highlighting and compilation only requiring one installation.

An initial client implementation was using a parser generator, PEG.js, but a 'from scratch' implementation turned out to be much less complex than the corresponding grammar. This may be because the 'from scratch' implementation is using a method of normalizing/simplifying the text before aplying regular expressions to validate the syntax, and/or because PEG.js' syntax for defining grammars may be more suitable for languages where line breaks and indentation do not carry meaning.

The client communicates with the audio generation system by OSC. It is not possible to use a parsed JavaScript data structure from within Csound so the data structure is encoded into Csound 'f-tables', i.e. the client will send OSC messages containing Csound 'scores' containing f-tables. Using f-tables instead of a nested data structure with named properties is very cumbersome, but seems to be necessary in the context of Csound.

Implementing the audio generation system with Csound was motivated by the following.

1. Higher level programmability via a textual interface.
2. Full support on multiple platforms, OS X, Windows, Linux, etc. (including ASIO support on Windows).
3. High performance even on older machines.
4. VST support through Michael Gogins' version of Csound.
5. Easy use of the implementation (one command line starts the program).

7. USAGE

Please refer to https://github.com/sorenjakobsen/maccs which contains the latest Macs/s implementation and instructions on its usage.

8. USABILITY

[A preliminary usability test involving a number of composers shall be made.]

9. RELATED WORK

Perhaps the only well-known, publically available tool which seems to share some design visions with Macs/s is TidalCycles by Alex McLean. Some similarities are

1. a text editing software based interface to support a creative music writing process.
2. support for updating playing music without breaking the flow of the music ('live coding').
3. limited capability in relation to sound design.

In contrast Macs/s

1. is more limited in capability and designed specifically for a method of 'modular' composition.
2. does not suggest using text editing software in a performance situation.

10. CONCLUSIONS

Macs/s has been presented and explained in relation to some ideas about music. Some assumptions throughout the article should be examined further. It remains to been seen which amount of impact on artistic practices Macs/s might have.

11. REFERENCES

[1] R. Gibb. The aesthetic of machinery: Objekt interviewed, 2013.

[2] R. Henke. Ableton live.

[3] R. Henke. Give me limits! two perspectives on computer based music production in a time of exponentially growing possibilities, 2016.

[4] J. Klug. Performance state, 2014.

[5] A. McLean. Making programming languages to dance to: Live coding with tidal. In *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design*, pages 63–70. ACM Sigplan, 2014.

[6] A. Monrad. Interactive/audiovisual projects.

[7] MusicTech. Squarepusher interview - brutal sound, 2015.

[8] P. Naur. *Computing: A Human Activity.* ACM Press/Addison-Wesley, New York, 1992.

[9] P. Tingen. Squarepusher, 2011.

[10] G. Toussaint. The euclidean algorithm generates traditional musical rhythms. In *Proceedings of BRIDGES: Mathematical Connections in Art, Music, and Science*, pages 47–56. The Bridges Organization, July - August 2005.

[11] G. Wang. Keynote: The me in nime, 2017.