DEVELOPING METHODS FOR COMPUTER PROGRAMMING BY MUSICAL PERFORMANCE AND COMPOSITION

Alexis Kirke
Interdisciplinary Centre for Computer Music Research,
University of Plymouth, UK

Eduardo R. Miranda
Interdisciplinary Centre for Computer Music Research,
University of Plymouth, UK

ABSTRACT

There has been much successful work in sonifying computer program code to help debugging. This paper investigates the reverse process, allowing music to be used to write computer programs. Such an approach would be less language dependent; would allow a more natural method of programming for affective computing; would provide a natural sonification of the program for debugging; includes the possibility of hands-free programming by whistling or humming; may help those with accessibility issues; and would help to mitigate one element that has consistently divided the the computer music community – those who can program and those who can't. This paper will propose methodologies for programming by music and describe approaches for investigating their utility.

1. INTRODUCTION

Recently there has been significant research in a number of tools for utilizing music to debug programs [1][2], as well as developer environments for non-sighted programmers [3]. These are based on the concept of sonification [4], i.e. turning non-musical data in musical data to aid its manipulation or understanding. In this paper we are interested in the reverse process, the use of music to generate non-musical data and processes, or de-sonification. In this paper the application of desonification to computer programming is investigated.

Musical structure has certain elements in common with program structure – this is one reason it can be used to sonify to help programmers debug. Music and programs are made up of modules and submodules. A program is made up of code lines, which build up into indented sections of code, which build up into modules. These modules are called by other modules and so forth, up to the top level of program. The top or middle levels of the program will often utilize modules multiple time, but with slight variations. Music is made up of multiple sections, each of which contain certain themes, some of which are repeated. The themes are made up of phrases which are sometimes used repeatedly, but with slight variations. (Also just as programmers re-use modules, so musicians re-use and "quote" phrases.) As well as having similarities to program structure, music contains another form of less explicit structure – an emotional structure. Music has often been described as a language of emotions [5]. It has been shown that affective states (emotions) play a vital role in human cognitive processing and expression [6]. As a result affective state processing has been incorporated into artificial intelligence processing and robotics [7]. The issue of writing programs with affective intelligence may be productively addressed using desonification. Other reasons to investigate a musical approach to programming include that fact it would be less language dependent; would provide a natural sonification of the program for debugging tools which have already been shown to be successful; includes the possibility of hands-free programming by whistling or humming; may help those with accessibility issues; and would help to mitigate one element that has consistently divided the musical community into from the computer music community – those who can program and those who can't.

Software desonification is related to the field of Natural Programming [8] – the search for more natural end-user software engineering. There is also a relationship between software desonification software and constraint-based, model-based and automated programming techniques. A musical approach to programming would certainly be more natural to non-technically-trained composers/performers who want to use computers, but it may also help to make computer programming more accessible to those who are normally nervous of interacting with software development environments. The use of humming or whistling methods may help to open up programming to many more people. If such an approach seems unnatural, imagine what the first QWERTY keyboard must have seemed like to most people. What would once have seemed like an unnatural approach is now fully absorbed into our society. In this paper we will start by examining a structured-based approach to programming with music. There will then be some initial discussion of what sort of interfaces might be required to allow for practical programming by desonification. This will be followed by an examination of other approaches to software desonification, and concluded with descriptions of future work.

2. STRUCTURE-BASED DESONIFICATION FOR PROGRAMMING

Because this topic has not been studied as far as we're aware, the development of a generalized theoretical approach is beyond the scope of this paper. Hence we will use the approach of giving examples to explicate some key issues. Musical structure is often described using a letter notation. For example if a piece of music

has a section, then a different section, then a repeat of the first section, it can be written as ABA. If the piece of music consists of the section A, followed by B and then a variation on A, it can be written as ABA'. Some forms in music are:

- Strophic - AAAA...
- Medley – e.g. ABCD..., AABBCCDD..., ABCD...A', AA'A''A'''A''''
- Binary – e.g AB, AABB...
- Ternary – e.g. ABA, AABA
- Rondo - , e.g. ABACADAEA, ABACABA, AA'BA''CA'''BA'''', ABA'CA'B'A
- Arch - ABCBA

There has been a significant amount of work into systems for automated analysis of music structure – e.g. [9] - though it is by no means a solved problem.

Suppose a piece of music has the very simple form in three sections ABA'. A is made up of a series of phrases and is followed by another set of phrases (some perhaps developing the motifs from the phrases in B), and A' is a transformed recapitulation of the phrases in A. Next suppose the sections A, B can be broken down into themes:

A = [*xy*]
B = [*eff*]

So A is a theme x, followed by a theme y. And B is a theme e followed by the theme f repeated twice. How might this represent a program structure? The first stage of a possible translation is shown in Figure 1.

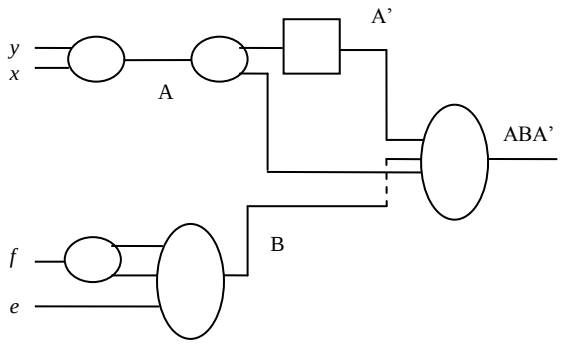

Figure 1. Graphical Representation of Structure

Each shape is an operation – the elliptical shapes represent ordered combinations (as in A is a combination of x and y in that order) or de-combinations (as in B includes *two* f's). Squares are transformations of some sort. Many mappings are possible, for example suppose that the combiners represent addition, the decombiners division by two, and the transformation is the sine function. Then the piece of music would represent a program:

$$Output = sin((x+y)/2) + (x+y)/2 + f/2 + f/2 + e$$

However such a specific mapping is of limited general utility. Removing such specific mappings, and returning to the abstraction based on the structure in Figure 1, and turning it into pseudocode could give something like Figure 2.

```
Def functionP(v1,v2,v3)
        //todo
End
Def functionQ(v1)
        //todo
End
Def functionR(v1, v2)
        //todo
End
        //
Def result = Main(x,y,e,f)
        Def A as var;
        Def B as var;
        Def Ap as var;
        A = functionP(x,y);
        [C,D] = functionQ(f);
        B = functionP(e,C,D);
        [E,F] = functionQ(A);
        Ap = functionR(E);
        Result= functionP(Ap,B,F);
end
```

Figure 2. Pseudocode Representation of ABA' piece.

The simplest way to explicate how this program relates to the musical structure is to re-draw Figure 1 in terms of the code notation. This is done in Figure 3. The multi-input ellipses are *functionP()*, the multi-output ellipses are *functionQ()*, and the square is *functionR()*.

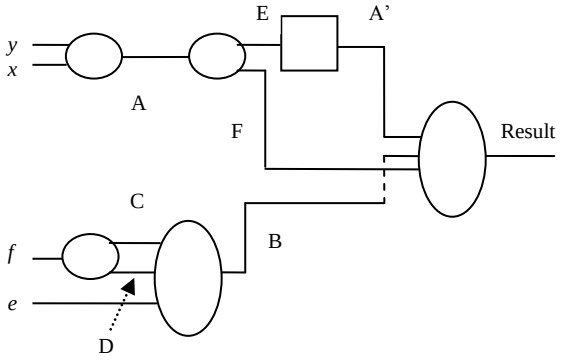

Figure 3. Figure 1 adjusted to explicate code in Figure 2

A few observations to make about this generated code are that even at this level of abstraction it is not a unique mapping of the music of of the Graphical Representation of the music in Figure 1. Furthermore it doesn't actually detail any algorithms – it is structure based, with "todo"s where the algorithm details are to be inserted. The question of how the todos could be filled in is actually partially implicit in the diagrams shown so far. The translation from A to A' would usually be done in a way which is recognizable to human ears. And if it is recognizable to human ears it can often be described verbally, which in turn may be mappable to computer code of one sort or another. For example, suppose that the change is a raising or lowering of pitch by 4

semitones, or doubling the tempo of the motifs, or any of the well-known transformations from Serialist music. Mappings could be defined from these transformations onto computer code. For example pitch rise could be addition, tempo change multiplication, and in the case of matrices (for example in Matlab programming) the mapping matrix from A to A' could be calculated using inverse techniques.

The transformation ideas highlight the approach of potentially utilizing the graphical form of the program mapping; i.e. programming using the MAX/MSP or Simulink approach. These methods are sometimes used by people who have limited software programming knowledge. So rather than converting the musical performance into textual code, such users may prefer to work with the graphical mappings. Such graphical mappings may suggest different emphases in approach to those found in mappings to text code in Figure 3. In the MAX-type case the role of motifs as "place-holders" is emphasized and the transformations performed on the motifs become the key constructive elements.) Also the graphical methods provide a possible real-time approach to programming. It would be easier for a user to see a steadily building graphical program while they play than to see the textual code. As can be seen from the above, part of the problem with investigating desonification for software is the number of possible mappings which could be defined. And given that a desonification approach to programming is so novel, testing different mappings is a complex process. Users would first need to become comfortable with the concept of programming by music, then we could compare different mapping techniques. But for them to become comfortable with the techniques the techniques to have been defined in the first place. Finding a likely set of techniques to test is a key problem for future work.

3. IMPLEMENTATION

How to implement musical programming is an open issue. Discussing the CAITLIN musical debugging system, [1] describes how "Ultimately, we hope the sound and light displays of multimodal programming systems will be standard items in the programmer's toolbox" and that "that combining auditory and visual external representations of programs will lead to new and improved ways of understanding and manipulating code." Another audio-based system – WAD – has actually been integrated into MS Visual Studio [3]. One way of implementing desonification for programming would be in a musically interactive environment. The user would see the program and hear a sonification of the program behavior or structure; they would then be able to play along on a MIDI keyboard or hum/whistle into a microphone to adjust the structure/behavior. One issue with this is that sonfication for audible display of program structure is a different problem. The types of musical mappings which best communicate a program structure to a user, may not be the best types of musical mappings which allow a user to manipulate the structure. A compromise would need to be investigated.

In the case of textual programming languages it may be necessary to actually sonify the structure as the user develops it – then the programming process would be that of the user "jamming" with the sonification to finalise the structure. For desonification for graphical programming, it may be simplest to just display the graphical modular structure and mappings generated by the music in realtime as the music is being performed by the user and analysed by the desonifier. One obvious point with any environment like this is it would require some practice and training. The issues of these environments is obviously key to the utility of software desonification, and are issues that are currently being worked on.

4. OTHER POSSIBLE DESONIFICATION ELEMENTS FOR PROGRAMMING

Another feature which music encodes is emotion. There has been some work on systems [10][11] which take as input a piece of music and estimate the emotion communicated by the music – e.g. is it happy, sad, angry etc. Affective state processing has been incorporated into artificial intelligence processing and robotics. It may be that the issue of writing programs with affective intelligence can be productively addressed using software desonification. Emotions in artificial intelligence programming are often seen as a form of underlying motivational subsystem[12]. So one approach to software desonification would be to generate the program structure using similar ideas already discussion in the structure section earlier, and then to generate a motivational subsystem through the emotional content of the music. Obviously this would require a different type of awareness in the musician/programmer; and it is not clear how any "if…then" type structure could emerge from the emotive layer only. One way might be as follows. There has been research which demonstrates that music can be viewed dramatically, with "characters" emerging and interacting [13]. This could provide a structure for which to map onto a multi-agent based programming environment, where agents have an affective/motivational substructure.

Musical programming could potentially be used in helping to create more believable software for affective computing, for example chat agents. Well-written music is good at communicating emotions, and communicating them in an *order* that seem natural, attractive or *logical*. This logic of emotion ordering could be used to program. Suppose a software programming environment utilizes one of the algorithms for detecting emotion in music. If a number of well known pieces of music are fed into the programming environment then in each piece of music it would auto-detect the emotional evolution. These evolutions could be transferred as, for example, markov models into an agent. Then the agent

could move between emotional states when communicating with users, based on the probability distribution of emotional evolution in human-written music. So the music is being used as emotional prototypes to program the agent affective-layer. One aspect of musical performance which linear code text cannot emulate is the ability for multiple performers to play at the same time and produce coherent and structured music. Whether this has any application in collaborative program could be investigated.

5. CONCLUSIONS AND FUTURE WORK

In this paper the reverse process of software sonification have been discussed, the use of music to write computer programs. Such an approach would be less language dependent; would allow a more natural method of programming for affective computing; would provide a natural sonification of the program for debugging; includes the possibility of hands-free programming by whistling or humming; may help those with accessibility issues; and would help to mitigate one element that has consistently divided the musical community into from the computer music community – those who can program and those who can't. This paper has shown how a simple musical structure could be mapped on to program code elements. This raises two questions: would it not be quicker to simply code the structure, if the user has any coding knowledge? Also if the user has no software knowledge, and generates the structure using music performance, surely they're still stuck because they can't fill in the gaps?

Neither can be fully answered without further research. But question 1 can be partially addressed through the observations about affective programming, and affective prototyping, as well as the possibility of merging the musical approach with program sonification methods. Answering Question 2 depends on more research into the application of program desonification to graphical programming, and the transformation of musical mappings into code detail (e.g. into mathematical symbols). The key development now required is to investigate the utility of real-time implementations of the approach, together with which mappings are most natural. The plan is to develop a real-time musical structure analysis tool, and from this two types of environments: (1) a real-time graphical programming environment which takes musical input; and (2) a real-time textual programming environment where the generated code structure is simultaneously sonified. For each of these a series of mappings will be designed to turn some standard musical structures into software code and structures. The test basis will be users of three types: musical experts with no software knowledge, users trained in music and software development, and users with no music or software training. For the third class a simple pitch recognition system will be used to allow them to hum, whistle or sing to generate program structure. Based on this a series of suitable mappings will be documented, together with their utility for the different types of user, and different types of programming tasks.

8. REFERENCES

[1] Vickers, P., Alty, J., "Siren songs and swan songs debugging with music", *Communications of the ACM*, v.46 n.7, July 2003

[2] Boccuzzo S., Gall H., "CocoViz with ambient audio software exploration", *ISCE'09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, IEEE Computer Society: Washington DC, U.S.A., 571–574, 2009

[3] Stefik, A., "On the Design of Program Execution Environments for Non-sighted Computer Programmers", PhD thesis, Washington State University, 2008.

[4] Cohen, J., "Monitoring Background Activities", In *Auditory Display: Sonification, Audification, and Auditory Interfaces*. Addison-Wesley, MA, 1994

[5] Cooke, D., *The Language of Music*. Oxford University Press, UK, 1959

[6] Malatesa, L., Karpouzis, K., Raouzaiou, A., "Affective intelligence: the human face of AI", In *Artificial intelligence*, Springer-Verlag Berlin, Heidelberg 2009

[7] Banik, S., Watanabe, K., Habib, M., Izumi, K., "Affection Based Multi-robot Team Work", *Lecture Notes in Electrical Engineering*, Volume 21, Part VIII, 355-375, 2008

[8] Myers, B. A., Ko, A., "More Natural and Open User Interface Tools", *Proceedings of the Workshop on the Future of User Interface Design Tools*, ACM Conference on Human Factors in Computing Systems, 2005

[9] Paulus, J., Klapuri, A., "Music structure analysis by finding repeated parts", *Proceedings of AMCMM 2006*, ACM, New York, USA 2006

[10] Kirke, A., Miranda, E. R. "A Biophysically Constrained Multi-Agent Systems Approach to Algorithmic Composition with Expressive Performance", *A-Life for Music*, A-R Editions, USA 2011

[11] Friberg, A., "A fuzzy analyzer of emotional expression in music performance and body motion", *Proceedings of Music and Music Science*, Stockholm, Sweden, 2004

[12] Izumi, K., Banik, S., Watanabe, K., "Behavior Generation in Robots by Emotional Motivation", *Proceedings of ISlE 2009*, Seoul, Korea 2009

[13] Maus, F., "Music as Drama", In *Music Theory Spectrum*, Vol. 10, Spring, University of California Press, 1988