Interacting with Text and Music: Exploring Tangible Augmentations to the Live Coding Interface

Chris Kiefer
Department of Informatics, University of Sussex, Brighton, UK, BN1 9QJ.
c.kiefer@sussex.ac.uk

Abstract. This project is an exploration of musical expression in relation to live coding. It examines the way in which performers interact with code, and highlights some tensions and dualisms that exist when interacting with live coding systems. The notion of *approximate programming* is proposed; where the performer can interactively manipulate and explore algorithms with a multiparametric controller, whilst retaining the code as the medium and the coding environment as the key focus. This system is trialed with a new interface, *Inflorescense*, highlighting some design challenges for this type of instrument.

Keywords. Live coding, multiparametric control, tangible user interfaces, musician-computer interaction, genetic programming.

1. Introduction

Writing as a musician who composes and performs with code as a matter of preference to other mediums, creating music through programming can be a very liberating experience but it can also be genuinely frustrating. Code gives the musician the freedom and power to engage very closely with digital sound synthesis processes, to create constructs and abstractions that relate to their own musical style and thought processes, less encumbered or constrained by preset and prescribed pattern, assumptions and structures. Code puts the musician at the bare interface to the sound synthesis process, where the points of engagement are the programming languages, and the interactive system used to manipulate the language. This is a huge space of musical possibilities, although with a heavy reliance on the skill of programming and understanding of digital music theory to effectively realise ideas. Code can also draw the musician away from the instantaneous and engaged interactive loop that is conventionally associated with playing a musical instrument and with musical expression. This creates a dissonance where the coder-musician has the power and flexibility to engage with musical process and realise ideas, but these ideas can't always be created in a musically expressive way.

Whether coded music should be played expressively is of course a matter of opinion and personal preference of the musician or performer. Part of the essence of live coding (A. Ward et al. 2004; N. Collins et al. 2003) is to engage with and manipulate the algorithm, in whatever of many forms this may take. Also, as Magnusson and Mendieta (2007) and Bertelsen et al. (2009) have shown, musicians enjoy engaging with resistances and limitations in musical systems, whether these are part of a physical system or whether they are conceptual resistances as you might find in the design of a programming language. There's no such thing as a perfect instrument, even a broken toy can sound good and be engaging to interact with. With these issues acknowledged, it's still pertinent to ask the question, are there ways of writing and manipulating code that are more musically expressive that current conventional methods? Are there ways to code musical processes that allow the programmer to directly and intuitively realise their ideas while still retaining the inherent advantages of the medium of code. To investigate this question one can focus on both the language and the way we manipulate it. With so many varying programming languages used for musical programming, and with much less variance in the manner in which we typically interact with code, it is this intention of this paper to be language agnostic, and focus on the aspects of musical coding concerning human-machine interactivity. This project sits firmly within

the context of live coding practice, focusing on dynamically interpreted programming languages being used performatively for music composition, performance and improvisation.

The paper begins by examining the conventional live coding interface, focusing on how the process of musical coding matches with the affordances of the laptop and text editor. A key question is asked: how precise does the act of coding need to be? And is this task context dependent? When programming a device driver or nuclear power plant safety system, precision and reliability is paramount. In music, expression rather than fine precision may often be an overriding priority. In this case, is it possible to use new interactive methods to produce code more expressively at the expense of precision? The concept of *approximate programming* is introduced, and a tangible interface (Ishii and Ullmer 1997), *Inflorescense*, is used in tandem with a conventional laptop interface to create and manipulate code, with novel expressive possibilities whilst retaining code as the medium. This setup is used in three experimental scenarios that are evaluated from an autoethnographic perspective (Magnusson 2011). This raises cognitive issues concerning production and comprehension of code, which are addressed in the discussion.

2. Related Work

Several projects have explored the use of Tangible User Interfaces (TUIs) for programming. Blackwell (Blackwell 2003) provides a theoretic analysis of the possibilities in this space. Horn and Jacob (2007) and Sapoundis and Demetriadis (2011) show examples of implemented systems. A musical example is *AudioCubes* (Schiettecatte and Vanderdonckt 2008). These systems typically use movement and linking of objects to configure programming logic, and are self-contained. This project's focus varies from these projects by attempting to augment rather than replace the coding environment, and by using continuous instead of discrete control. Biegel et al. (2014) and Raab et al's (2013) *RefactorPad* explore augmentation of programming IDEs with gestural control using multitouch interfaces. Both studies find this approach to be promising. There's some work in this area specific to live coding. Collins and Blackwell (2005) focus on the programming language as a musical instrument, outlining the task demands of live coding interfaces. McLean et al. (2010) explore how we perceive code and stress that techniques for augmenting code with visual cues are central to live coding and effective comprehension of code. In terms of interaction with computers, Armstrong (2006) studies the interactive possibilities offered by conventional computers and argues that they preclude the embodied modes of interaction preferred by musicians. Magnusson (2009) provides a detailed analysis of how we interact with screen based musical instruments.

3. Interacting with Code

This section analyses the interactive possibilities of conventional live coding systems, and argues that an understanding of time within live coding is key to understanding musical expression through code. This paper considers a conventional setup to be a standard laptop. It is acknowledged that many performers extend their computer with different controllers, which are typically mapped to musical parameters (which may have been set up using code). This study focuses on augmentations to the coding environment rather than control of musical parameters, and so does not take this type of control into consideration.

McLean (2014) proposes three live feedback loops that are created in live coding performances: (1) manipulation feedback, between performer and code, (2) performance feedback between the performer and music and (3) social feedback between performer and audience. These loops are arguably co-dependent with common paths, and can be combined as in Figure 1. The shared paths within these feedback loops highlight a tension that exists in live coding systems; the system must fulfil three functions: musical performance, musical creation and support for programming. The tension between the requirements of the instrument for fulfilling these functions can preclude musical expression. Another important factor affecting musical expression is timely interaction. Armstrong (2006) presents an analysis of the *computer-as-it-comes* (which we can consider to be the standard setup for live coding) from an enactive perspective. He finds a disconnection where the *computer-as-it-comes* precludes embodied modes of interaction normally associated with musical expression. He presents a set of five criteria for

embodied modes of interaction, two of which are especially relevant for discussion in the context of live coding. Firstly, embodied activity is situated; the performer must be able to adapt to changes in the environment without full prior knowledge of the environment. Secondly, embodied activity is timely; the performer must have a capacity for action within real-time constraints. These requirements can be seen as co-dependent; it would be ideal if the performer could react to the audience or other musicians in a timely manner. The expectation of what is timely in a live coding context may not be the same as for musicians playing controllers or acoustic style instruments, however there's no doubt that it's can be difficult to code large structures and make large changes quickly. To analyse this further, we need to examine the concept of time in live coding performance, and take a closer look at how the interface affects the speed of interaction.

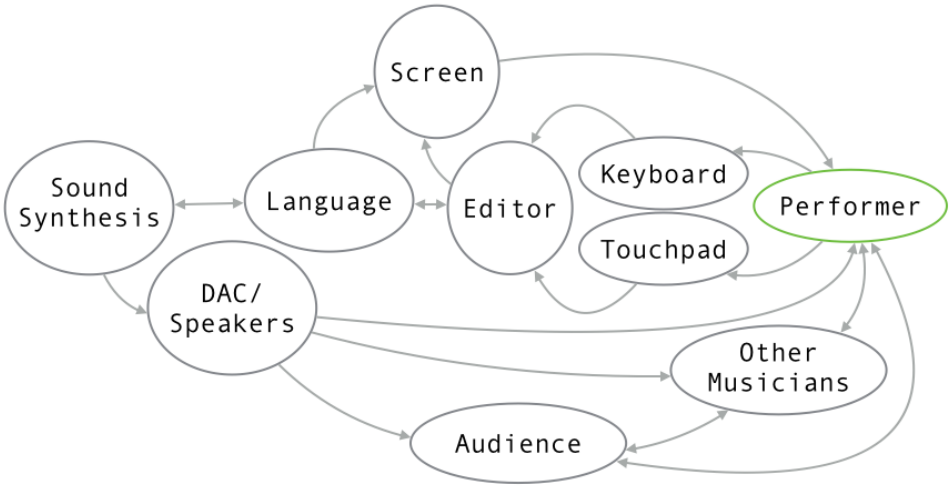

Fig. 1. Manipulative, Performative and Social Feedback Loops

The live coding performer must move in and out of real-time, and this is consistent with the act of live coding which is to manipulate a running system rather than directly play it. In fact the performer may very rarely or maybe never directly trigger a musical event in realtime. Many performance activities may involve tasks that move further and further away from the possibility of being conducted in musical time, for example queueing events to run quantise points, editing code before compiling it, writing new code structures, debugging, planning structural changes and so on. Despite this variation in time constraints for different tasks, there can still be an overall pressure to react in a timely manner, to be able to express musical ideas in musical time. This highlights another dualism within the live coding setup. Many features of live coding systems are partial solutions to this issue of timely interaction with code, primarily the use of dynamic programming languages but also features of IDEs and text editors that allow fast and intuitive access to functional code management tasks such as selection and execution, navigation within and between documents, and code visualisation techniques that act as cognitive supports for the programmer.

Putting these threads together, two dualisms are highlighted in the live coding instrument, which must try to satisfy competing requirements. While it must support the performer in interacting with musicians and audience members in the external environment, it must also support interaction between the performer and programming IDE. While it would ideally allow the performer to interact with the audience in a timely manner, in musical time, it does this through a system with a range of functionalities and time constraints, mostly occurring in non-musical time. The next section asks if there are possible solutions to bridging this variation in time constraints.

4. Approximate Programming

This section proposes the notion of *approximate programming*, an exploratory method for producing code quickly to satisfy musical time constraints at the cost of losing accuracy. One possible way to program more quickly is to use a high level process to produce code, which approximately satisfies a requirement. If the process can generate an approximate solution, then this can be fine-tuned by the coder; this would provide a method for creating large structures quickly according broad requirements. The notion of an approximate program is somewhat incongruous to conventional computing scenarios; many computing tasks such as safety critical systems require very high accuracy and testability. Musical improvisation has a different set of constraints where accuracy need not always be so important; this is especially relevant in electronic music where, given the complexity and variation of sound synthesis processes, the musician cannot always accurately predict how a change in code will change the sound. Electronic music performance may take the form of an evaluative loop where the performer experiments with new synthesis processes in an iterative manner, so approximation is already built into this process. Assuming that the constraint of accuracy can be relaxed in musical coding, how could the process of approximate programming work? The system proposed here takes continuous input from a multiparametric (Kiefer 2012) controller, converts this into code based on a set of prewritten component functions, and plays the result in realtime. The use of a physical controller and realtime synthesis allows musical-time interaction in the style of a conventional musical controller, although the performer is interacting through the medium of code rather than a mapping engine to synthesis or sequencing parameters. The code medium puts this system in a hybrid domain of musical controller and code editor where the performer can interact at different levels of abstraction.

To convert a set of continuous parameter streams into code, the system borrows representations from the field of genetic programming (GP) (Poli, Langdon, and McPhee 2008). There has been a small amount of previous work in this area, for example Poli and Cagnoni (1997) used interactive GP to search for image processing algorithms. The difference in this project is that the code is retained as the medium; this is an extension to the IDE and interaction with the code remains central to the search process. GP researchers have developed numerical representations for algorithms, and methods for translating these representations into code, so that solutions to tasks can be discovered though evolutionary methods. In GP, an algorithm is represented as a tree of functions, and the choice of functions and their parameters is determined by a set of numbers in a gene. The system in this project works as follows: a set of component functions is decided upon that will be *ingredients* for the algorithm being generated. The system takes a set of floating point numbers, P, as input. The first number is used as an index into the set of component functions, to determine the function at the top of the tree. The next numbers in P are used as constant parameters for this function. When constants are added, a list of their locations in the tree is maintained. A new number is taken from P, and used as an index to choose a constant, C, that will be replaced with a new function. The value of C determines the function that replaces it, and the next numbers from P determine this function's constant parameters. This process continues while more numbers are available in P, thereby building a tree of functions that represent an algorithm

In use, the multiparametric controller generates multiple streams of numbers, and the system takes snapshots of these and converts them to a code tree, which is then evaluated and run as a DSP process at audio rate. The system is implemented in SuperCollider, and the algorithms are built as *SynthDefs*. The set of component functions can be modified or live coded while using the code generator, allowing the performer to shape the search space. For example, to try and find FM-style sounds, the component functions could be multiplication, addition and sine wave generation. To generate rhythmic sounds, the component functions could be delays, impulses and and resonators. This system allows the performer to specify a high-level concept through component functions and then explore this by using a physical controller and interacting with the code editor, in order to find an approximate solution. This system is controller agnostic; any multiparametric controller will work with it, given that the parameter count is large enough to represent the complexity of the desired outcome. To explore the use of the system, it was paired with a new controller, *Inflorescense*, and trialed in a number of scenarios.

5. Inflorescense

Inflorescence is a modular multiparametric controller, which is based around the form of a plant. It consists of a number of stalks that end with a flower, housing a motion sensor and a multicolour LED. The stalks are made from braided armature wire, which keeps its shape when deformed. This allows the motions sensors to be moved into different configurations and formations. Up to sixteen stalks can be used in the instrument, with each one sending out measurements from a 3-axis accelerometer and a 3-axis gyroscope. A microcontroller board collects data from the stalks, and controls the LED colours. The board connects to SuperCollider via a serial interface. The design of *Inflorenscense* aims to be organic and uneven, with varying shaped stalks that affect each other's movement. This makes it a compelling interface and research tool for exploring approximate programming; it embodies opposing interactive qualities to the *computer-as-it-comes*: imprecision, sensitive continuous control and gestural interaction. Three scenarios were designed to test the controller with approximate programming, with particular emphasis on interacting with the code and code editor: (1) a basic scenario where code was generated with a static set of component functions, and visual inspection of the code guided interaction, (2) the component functions were live coded while using the interface and (3) a mechanism was built where a subsection of the generated code could be selected and manipulated by the controller, allowing selective editing of the function tree.

5.1 Observations

The use of a controller for code generation gives an interesting option of either ignoring the code and treating the controller as a self-contained instrument, or using the controller as an extension to the code. It's the author's opinion that keeping the code as the central focus adds significant value to the system. The interplay between visual inspection of the generated algorithm and movement of the controller is an engaging process of discovery. With the addition of coding the component functions, it's possible to move the algorithm into an approximate target area, and selection of code in scenario (3) enables a fine-tuning process that still uses the approximation system. There are a few challenges in use. The way the algorithm is encoded from the GP representation is quite non-linear in places and very linear in others, so the mappings can be unintuitive in some areas. Also, the system requires fast comprehension of the generated algorithm, which can be difficult; visual cues are a huge help in this respect.

6. Conclusions

This paper has analysed the way in which live coding performers interact with conventional live coding systems, and highlighted the tension between interaction in musical time and non-musical time. Approximate programming has been suggested as a method for creating large but imprecise code structures in a timely and embodied way. In use, the system seems promising, although there are some design challenges to overcoming concerning code comprehension and numerical representation.

References

Armstrong, Newton. 2006. "An Enactive Approach to Digital Musical Instrument Design." PhD thesis, Princeton University.

Bertelsen, Olav W., Morten Breinbjerg, and Søren Pold. 2009. "Emerging Materiality: Reflections on Creative Use of Software in Electronic Music Composition." *Leonardo* 42 (3): 97–202.

Biegel, Benjamin, Julien Hoffmann, Artur Lipinski, and Stephan Diehl. 2014. "U Can Touch This: Touchifying an IDE." In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, 8–15. CHASE 2014. New York, NY, USA: ACM.

Blackwell, Alan. 2003. "Cognitive Dimensions of Tangible Programming Techniques." In *Proceedings of the First Joint Conference of EASE and PPIG*, 391–405.

Blackwell, Alan, and Nick Collins. 2005. “The Programming Language as a Musical Instrument.” In *In Proceedings of PPIG05 (Psychology of Programming Interest Group*, 120–30.

Collins, N., A. McLean, J. Rohrhuber, and A. Ward. 2003. “Live Coding in Laptop Performance.” *Organised Sound*.

Horn, Michael S, and Robert JK Jacob. 2007. “Designing Tangible Programming Languages for Classroom Use.” In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, 159–62. ACM.

Ishii, Hiroshi, and Brygg Ullmer. 1997. “Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms.” In *CHI ’97: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

Kiefer, Chris. 2012. “Multiparametric Interfaces for Fine-Grained Control of Digital Music.” PhD thesis, University of Sussex.

Magnusson, Thor. 2009. “Epistemic Tools: The Phenomenology of Musical Instruments.” PhD thesis, University of Sussex.

Magnusson, Thor. 2011. “Confessions of a Live Coder.” In *Proceedings of the International Computer Music Conference*. ICMA.

Magnusson, Thor, and Enrike Hurtado Mendieta. 2007. “The Acoustic, the Digital and the Body: a Survey on Musical Instruments.” In *NIME ’07: Proceedings of the 7th International Conference on New Interfaces for Musical Expression*, 94–99. New York, NY, USA: ACM.

McLean, Alex. 2014. “Making programming languages to dance to: live coding with tidal”. Proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design.

McLean, Alex, Dave Griffiths, Nick Collins, and G Wiggins. 2010. “Visualisation of Live Code.” In *Electronic Visualisation and the Arts*.

Poli, Riccardo and Stefano Cagnoni. 1997. “Genetic Programming with User-Driven Selection: Experiments on the Evolution of Algorithms for Image Enhancement”. In *Genetic Programming: Proc. 2nd Annual Conf.*, 269-277.

Poli, Riccardo, William B. Langdon, and Nicholas Freitag McPhee. 2008. *A Field Guide to Genetic Programming*. Published via http://lulu.com; freely available at http://www.gp-field-guide.org.uk. http://dl.acm.org/citation.cfm?id=1796422.

Raab, Felix, Christian Wolff, and Florian Echtler. 2013. “Refactorpad: Editing Source Code on Touchscreens.” In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 223–28. ACM.

Sapounidis, Theodosios, and Stavros Demetriadis. 2011. “Touch Your Program with Hands: Qualities in Tangible Programming Tools for Novice.” In *Informatics (PCI), 2011 15th Panhellenic Conference on*, 363–67. IEEE.

Schiettecatte, Bert, and Jean Vanderdonckt. 2008. “AudioCubes: a Distributed Cube Tangible Interface Based on Interaction Range for Sound Design.” In *TEI ’08: Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*, 3–10. New York, NY, USA: ACM.

Ward, Adrian, Julian Rohrhuber, Fredrik Olofsson, Alex McLean, Dave Griffiths, Nick Collins, and Amy Alexander. 2004. “Live Algorithm Programming and a Temporary Organisation for Its Promotion.” In *Proceedings of the README Software Art Conference*, 243–61.