Performance Systems for Live Coders and Non-Coders

Avneesh Sarwate, Ryan Rose, Jason Freeman
Georgia Tech Center for Music Technology
840 McMillan Street NW
Atlanta, GA
{avneesh, rytrose, jason.freeman}@gatech.edu

Jack Armitage
Centre for Digital Music, Queen Mary U of London
Mile End Road London E1 4NS
London, United Kingdom
j.d.k.armitage@qmul.ac.uk

ABSTRACT

This paper explores the question of how live coding musicians can perform with musicians who are not using code (such as acoustic instrumentalists or those using graphical and tangible electronic interfaces). This paper investigates performance systems that facilitate improvisation where the musicians can interact not just by listening to each other and changing their own output, but also by manipulating the data stream of the other performer(s). In a course of performance-led research four prototypes were built and analyzed using concepts from NIME and creative collaboration literature. Based on this analysis it was found that the systems should 1) provide a commonly modifiable visual representation of musical data for both coder and non-coder, and 2) provide some independent means of sound production for each user, giving the non-coder the ability to slow down and make non-realtime decisions for greater performance flexibility.

Author Keywords

Live coding, creative collaboration, NIME design

CCS Concepts

•Applied computing → Performing arts; Sound and music computing;

1. INTRODUCTION

Live coding is an art form where performers create art through writing code in real time, often in front of an audience [5]. Our research focuses on the interplay of two collaborating parties: live coding musicians and non-coding musicians. By non-coding musicians, we mean any musician who does not write code during a performance. In our work, non-coders have used both traditional instrument forms (e.g. guitar, keyboard) and graphical and tangible interfaces (e.g. touch screen, MIDI controller, NIMEs). We seek to design interfaces where both live coders and non-coders can contribute and interoperate in a musical performance with satisfactory knowledge of their collaborator's actions.

To investigate interface designs that lead to successful collaboration, we conducted a program of performance-led research based on Benford et. al.'s existing methodology [1]. We developed and tested four performance interfaces (each consisting of a graphical/tangible interface for the non-coder and a live coding API in Python for the live coder), with each iteration being built off knowledge gained from performing with and analyzing the previous. Testing occurred in two forms: the authors improvised with the interfaces in an unrestricted, private setting, and also in a single public performance of semi-improvised music.

Benford et al. [1] corroborate the effectiveness of this research strategy, especially for interface design in artistic pursuits. Specifically, we employ this strategy with Benford's fifth relationship in mind: "Theory can also guide practice, distilling artists' 'craft knowledge' into guidelines that can be used by other artists or perhaps practitioners in other fields" [1]. Our interface development highlights recurring issues in performance systems combining live coders and non-coders, for which we present potential solutions. At the time of writing, this research is exploratory and part of a larger investigation into systems that incorporate both coding and non-coding interfaces. Our current analysis finds that a commonly modifiable visual representation of musical data for both coder and non-coder greatly increases both musicians ability to communicate musical change and make informed musical decisions. Also, providing some independent means of sound production for each user gives the non-coder the ability to slow down and make non-realtime decisions, which prevents them from being overloaded by complex output from the live coder.

This paper is structured as follows: each of our prototypes' motivations are described, followed by a description of the system and its use. After testing the system through personal performance, we then analyze the successes and failures using tools from creative collaboration literature. Lastly, we synthesize the trends in our analysis into design recommendations.

2. RELATED WORK

There have been multiple systems created that explore communication between coding and non-coding musicians. One such system is Magnussons' piece Fermata, which combines the live coded Threnoscope and an augmented bass clarinet [7]. However, Fermata has the Threnoscope and bass clarinet operating as independent systems coordinated only by human intention rather than by some formal mechanism. We seek to incorporate a formal computational relationship between the information the two parties input into a system. Another collaborative system is Eldridge et al.'s feedback cello, in which a self-resonating cello can be live coded, but there is a strict limitation on the flexibility of the collaboration due to this being a highly-specialized instrument, both in terms of musical aesthetics and the requirement of a physically resonant instrument body [6]. Other examples in live coding literature reference the ability to dynamically remap

NIME'18, June 3-6, 2018, Blacksburg, Virginia, USA.

interface parameters [2, 11], but are more focused on proving the potential of such dynamic mapping systems than analyzing the types of mappings possible. More broadly, Lee and Essl's live coded mobile music instrument successfully couples a live coder and an instrumentalist: the live-coder generates a touch-screen interface on which the non-coder performs [8]. This collaborative experience is desirable because interacting with symbolic abstractions of musical data in real time provides the potential for deeper interaction. We seek to model this experience, but plan to focus on developing a more static interface for the non-coder.

As part of our research we analyzed our prototypes with respect to concepts from NIME and creative collaboration literature. The first is the concept of mutual engagement applied to creative collaborations proposed by Bryan-Kinns et al [3]. They define mutual engagement as the point where collaborators are "engaged with both the product at hand and with others in the collaboration". Bryan-Kinns et al.'s work also provides design goals for facilitating mutual engagement. We chose this framework because it studies features of realtime collaboration in the creative domain [3]. We also used Nash and Blackwell's concept of liveness in musical interfaces to investigate how coders and non-coders operate at different time scales [9]. Nash defines level 3 liveness as "edit-triggered" systems, such as live coding, "in which edits to the representation instantly trigger feedback, allowing users to make rapid actions and (after system response) a chance to correct mistakes" [9]. Nash then defines level 4 liveness as "stream-driven", such as live performance, "in which the program is continually active, and where edits directly affect program execution, providing high visibility of the effect of actions" [9]. Nash's work provides a useful lens for examining how a live-coder and non-coder may communicate across timescales.

3. PROTOTYPES

3.1 Prototype I - Response Functions

For our first prototype we adapted an idea from Dan Tepfer's experiments with the Yamaha Disklavier. Tepfer would play a melody on his instrument, which would be then transformed by a computer program and played to him in real-time [4]. For Tepfer, the transformation programs were defined before the performance, and we were interested in exploring how live coding could be used to define these transformations in real-time.

Our hypothesis was that the non-coder and live coder would pass musical material back and forth, much like jazz musicians improvising by trading fours. The non-coder would seed the system with a melody, and its transformation by the response function would provide inspiration for further content, and this cyclical process would continue.

The transformation we experimented with was buffer shuffling, i.e. permutation of sliced segments of a melody (Figure 1). To give some control over this response to the non-coding musician, we developed two different triggers that the non-coder could use to call these response functions. The first trigger activated the response function when the non-coding musician held a note longer than a specified amount of time (set by live coder). The other trigger was a specified duration of silence (also set by the live coder). Only one type of trigger could be active at a time, determined by the live coder. To implement this system we used a Novation Launchpad as a "keyboard", which the non-coder would play on as a typical note-triggering MIDI interface.

In practice, the actions of the live coder proved too complex for the non-coder to decipher and respond to. The keyboard player (non-coder) couldn't decipher complex shuffings at all, and for simpler shufflings, they still couldn't understand them fast enough to respond quickly to changes in the pattern. In terms of Nash's levels of liveness, the live coder is engaging in a level 3 activity, and has time to produce complex output. The non-coder, as a level 4 real-time performer, is forced to parse the complex output of the live coder on the fly. Furthermore, the live coder can use the code as a visual aid when constructing the permutation. The non-coder has no visual reference at all. In terms of mutual engagement, this violates Bryan-Kinns et al.'s guideline that collaborators should share a consistent representation of the object being manipulated, in this case, music. These difficulties were exacerbated by the fact that a musical permutation is not a structure that the performers were musically familiar with, and thus had more trouble thinking about than common structures such as loops.

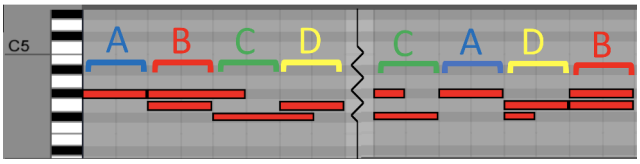

Figure 1: An example of buffer shuffling. The pattern `"ABCD"` becomes `"CADB"`.

3.2 Prototype II - Loop Sequencer

Inspired by the deficiencies of the previous system, we hypothesized that a system with clearer visualized musical data, represented with more familiar metaphors, would be more successful.

For this prototype's non-coding interface we implemented a step sequencer, where each step is a measure-long melody. These melodies are recorded in real-time via MIDI controller. The system provides an optional metronome track, to facilitate real-time melody recording. The step sequencer interface was a traditional toggle grid, implemented on a Novation Launchpad.

The live coder had the power to manipulate the sequencer grid as well as apply transformations to the recorded melodies. The transformations were the counterpoint transforms of inversion, retrograde, and transposition. With the ability to apply and undo these transformations at will, the live coder could contribute to the musical output of performances independent of the actions of the non-coder. This independence was lacking in the previous prototype.

We found performance with this system more engaging than with the previous. While performing we had better understanding of each other's actions, and could modify each other's musical output more effectively. Instead of the constant call-and-response of the previous prototype, this system provided a persistent structure (a melody loop) that did not require continuous activity to generate musical output. This allowed the non-coder to have both level 4 liveness activity (e.g. improvising on the keyboard over the sequenced melodies) and level 3 liveness activity (e.g. setting the sequencer arrangement, planning new melodies to record).

Furthermore, the visualization of a common representation led to well-informed decision-making by the performers. The step sequencer interface implemented Nielsen's heuristic of a highly visible system state, which clearly communicates what was happening musically [10]. Because both performers were working with the same representation and manipulating the same objects, modification of each other's output through the step sequencer was facilitated, meeting Bryan-Kinns et al.'s guideline of mutual modifiability. For

the performers, looping melodies were easier to reason with than abstract musical permutations.

However, the visualization did not extend to the live coder's transformation of the non-coder's melodies. While the persistence of the loop allowed the non-coder to parse the transformation aurally over multiple loop playbacks, this added avoidable processing time. Here, mutual modifiability on the loop level was not achieved, as the transformations were not removable by the non-coder. This could have been ameliorated through a simple undo/redo button for the non-coder.

3.3 Prototype III - Parameter Curves

Drawing on the success of visualization from the previous interface, we were motivated to create a system with *all* musical data visualized. Additionally, we wanted to explore how a live coder could interact with an audio signal from a more traditional instrument. This prototype introduces a parameter curve system, where the live coder could define curve functions of varying complexity, and map these curves to parameters on a signal processing effect chain through which the non-coder (an electric guitarist) is playing. Because curves can be easily visualized, we hypothesized that full exposure of state would lead to musically rich output.

The signal chain consisted of a grain delay, a conventional delay, a pitch shifting module, and reverb. In an attempt to not overwhelm the guitarist with too much parameter change, only two parameters of these effects were ever being dynamically manipulated at one time. We designed a visualization where at least one period of the curve function would be shown, and a tracking line would move across time to show the current level of the effect (Figure 2).

Figure 2: Curve visualization and code snippet from the Parameter Curves interface.

While our visual interface properly communicated relative effect values, this continuous-valued effects parameter notation was very non-standard and challenging for the guitarist to make sense of musically. Through this prototype's design process, we discovered that there is a key difference between being understandable as an interface, and being understandable in a musical sense. This is illuminated when examining this interface through the lens of liveness: much like the first interface, the live coder, operating at level 3 timescale, is creating a complex object (the curve envelope) that the guitarist must respond to at a level 4 timescale. In this instance, the object was not intuitively understood because parameter curves are not a structure that guitarists are familiar with. The usefulness of the visualization was thus lost, due to the type of information conveyed.

Furthermore, the interface violated the consistent representation feature of Bryan-Kinns' mutual engagement. The continuous parameter curve provided to the guitarist was incongruous with the discrete note data they were producing. A more successful system could have matched the data forms: displaying a stream of chords to a guitarist playing notes, or giving parameter curves to a musician controlling the knobs of a modular synthesizer.

Lastly, this prototype suffered from the same problem for the live coder as the Response Functions system: input from the non-coder was required for the live coder to affect output. Again, the non-coder struggled to generate material, which limited live coder contribution as well.

3.4 Prototype IV - Orbit Environment

For our final interface, we sought to fix issues from the previous systems by designing the graphical interface and live coding API in tandem rather than adding live coding to an existing, non-coded system. In addition, we wanted to experiment with a procedural system that would lend itself to a high degree of visibility for the non-coder. To meet these constraints, we implemented a live coded physics environment where sound was generated by moving balls passing through positional triggers. A combination of touch screen and MIDI controller input allowed the non-coder to adjust the trajectory of these balls, which traveled in periodic paths across the environment. The live coder positioned lines across the environment, which triggered a note when crossed by a ball. The musical mappings of the notes triggered were defined by the live coder. The non-coder could manipulate global effects (e.g. bit-crushing, filtering, delay) through a separate, smaller, ball environment. In this environment, each ball was mapped to a different parameter, and the value of the parameter was defined by the distance of the ball from the center of the environment (Figure 3).

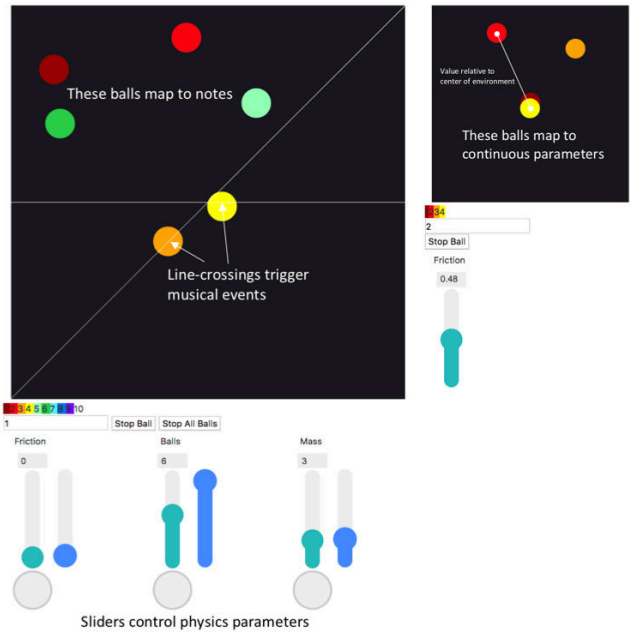

Figure 3: The Orbit Environment interface.

This system succeeded in giving the non-coder the flexibility to operate on level 3 or level 4 timescales, much like the Loop Sequencer, and allowed the coder to manipulate sound without being dependent on the level 4 actions of the non-coder. However, the system representation was musically non-standard and limited the ability of the performers to make sense of the music. For example, due to the geometry of the physics environment, it was difficult to predict the rhythmic paths of the balls, even though they were operating periodically.

The parameters of this system were not mutually editable. Though both parties had control over system parameters, they could not effectively control each other's parameters. This prevents even the mirroring of behavior between parties, a detriment to mutual engagement as noted by Bryan-Kinns et al. Furthermore, the physics environment's rules

were too complex to allow any action to have a fully predictable outcome. A performer is unable to influence another for two reasons: first, as mentioned, they don't have access to the other's parameters. Second, the system is too complex to compensate the actions of the other performer with their own. For example, if the non-coder set a ball trajectory and the live coder wanted to implement a specific change to the rhythm generated by that ball, the live coder would have to change a different parameter than the ball trajectory directly. The live coder would need to move the trigger line to compensate - an action too complex to apply accurately. This lack of mutual modifiability led to musically directionless performances. Finally, the system lacked visualization of the musical mappings for the non-coder. While the live coder could reference the code as was possible in the Response Function system, the non-coder only had aural access to the mappings. The result was a high memory load for the non-coder [10], which made the level 4 action of planning note triggers exceedingly difficult.

4. DISCUSSION

4.1 Common Design Challenges

Two main design issues revealed themselves repeatedly through the analysis of our prototypes. The first related to the translation of musical data: if the musical representation of the live coder's actions was not immediately understandable to the non-coder, formulating a musical response was difficult as the non-coder had to first process the representation. This surfaced in the response functions of the first prototype, the parameter curves of the third, and the physics parameters of the fourth. The second prototype avoided this issue of musical representation through an intuitive visualization of musical data via the loop sequencer.

The second issue arose from complications of the performers operating on different timescales. The live coder was often able to take time to produce complex musical output, which the non-coder was forced to respond to immediately for output of the system to continue. This was prevalent again in the struggles of the non-coder using prototype I and prototype III. Both the second and the fourth prototypes avoided this by making sound generation not directly dependent on level 4 liveness input from the non-coder (e.g. in prototype II loops playback without input, and in prototype IV balls move autonomously triggering notes).

Based on these common problems and instances of their mitigation, we propose the following two design guidelines.

4.2 Mutually Modifiable Visual Representations

Musical state should be represented visualized in a mutually modifiable interface with musical representations intuitively familiar to the performers (particularly the non-coder). Mutual modifiablity of a common representation allows both performers to directly modify the musical output of a collaborator, rather than being forced to "compensate" for the change in one set of parameters by changing a different set of parameters. Furthermore, selecting a representation that is familiar to both performers allows them to respond quickly to musical changes. Familiar representation is particularly important for the non-coder, as an unfamiliar representation could hamper their ability to informedly respond to musical change in real time.

4.3 Independent Sound Generation

The live coder and non-coder should have some independent methods for generating sound. By allowing the live coder to control the manipulation of sound without constant input from the non-coder, the non-coder is free to engage in non-real time activity, thus expanding the scope of actions that they can take during the performance. This also prevents either performer from being unable to generate music due to the inactivity of their collaborator.

5. REFERENCES

[1] S. Benford, C. Greenhalgh, A. Crabtree, M. Flintham, B. Walker, J. Marshall, B. Koleva, S. Rennick Egglestone, G. Giannachi, M. Adams, and others. Performance-led research in the wild. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 20(3):14, 2013.

[2] A. R. Brown and A. C. Sorensen. aa-cell in practice: An approach to musical live coding. In *Proceedings of the International Computer Music Conference*, pages 292–299. International Computer Music Association, 2007.

[3] N. Bryan-Kinns, P. Healey, and J. Leach. Exploring mutual engagement in creative collaborations. In *Proceedings of the 6th ACM SIGCHI conference on creativity & cognition*, C&C '07, pages 223–232. ACM, June 2007.

[4] N. Chinen. Fascinating Algorithm: Dan Tepfer's Player Piano Is His Composing Partner. `https://www.npr.org/2017/07/24/538677517/fascinating-algorithm-dan-tepfers-player-piano-is-his-composing-partner`, July 2017.

[5] N. Collins, A. McLean, J. Rohrhuber, and A. Ward. Live coding in laptop performance. *Organised sound*, 8(3):321–330, 2003.

[6] A. Eldridge and C. Kiefer. Self-resonating Feedback Cello: Interfacing gestural and generative processes in improvised performance. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 25–29, Copenhagen, Denmark, 2017. Aalborg University Copenhagen.

[7] P. Furniss. Live coding meets augmented instruments. `https://petefurniss.wordpress.com/2016/01/12/live-coding-meets-augmented-instruments/`, Jan. 2016.

[8] S. W. Lee and G. Essl. Live Coding The Mobile Music Instrument. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 493–498, Daejeon, Republic of Korea, May 2013. Graduate School of Culture Technology, KAIST.

[9] C. Nash and A. Blackwell. Liveness and Flow in Notation Use. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Ann Arbor, Michigan, 2012. University of Michigan.

[10] J. Nielsen. Heuristic evaluation. *Usability inspection methods*, 17(1):25–62, 1994.

[11] G. Wang, A. Misra, A. Kapur, and P. R. Cook. Yeah, ChucK it! dynamic, controllable interface mapping. In *Proceedings of the 2005 conference on New interfaces for musical expression*, pages 196–199. National University of Singapore, 2005.