Directions in Computational Music

Ian Clester
Center for Music Technology
Georgia Institute of Technology
ijc@gatech.edu

Abstract
This doctoral consortium submission describes my work on computational music, which has primarily focused on three problems: representation, distribution, and environment. Among other topics, this work relates to programming language design, interactive development, live coding, and programming interfaces. I welcome feedback from the PPIG community, especially regarding methodology and future directions, which I anticipate will prove helpful at this juncture midway through my doctoral program.

1. Introduction
Computational music is music represented as a computer program rather than as fixed media (such as a score, DAW project, MIDI file, or audio file). Its representation may thus be far more compact than its output (as it can directly encode abstractions of the composer's choice), and it may encompass many possible outputs by depending on external input or state (e.g. using a different seed for a random number generator). This notion is closely related to generative music (Eno, 1996), aleatoric music (Cage, 2004), process music (Reich, 1965), and algorithmic composition (Essl, 2007).

The general aim of my work is to realize the possibilities of computational music, primarily through the analysis and development of languages, tools, and interfaces for its creation and use. I have been pursuing this aim along three lines of investigation. The first is representation: how can computational compositions be written down, and which abstractions are useful or necessary for a computer music language? The second is distribution: how can computational compositions be effectively distributed (i.e. with ease approximating that of distributing static, recorded music)? The third is environment: what is a 'good' interface for creating computational compositions, and how can the possibilities enabled by computational composition be extended to composers who do not consider themselves programmers?

These questions relate to several topics of interest in the PPIG community. The question of representation relates to programming language design and choice of abstraction, as embodied in the variety of computer music languages and libraries available today. The question of distribution relates to issues of software development, dependency management, containerization, and virtualization. And the question of environment relates to user interfaces, interactive development, HCI, and programming pedagogy.

2. Existing & Related Work
I have conducted initial work in each of the three lines of investigation described above. The question of representation is explored in Aleatora (Clester & Freeman, 2021), a composition framework that challenges concepts taken for granted in most computer music systems. It is developed as a library for an existing, general-purpose programming language (Python), and it eschews hard distinctions between score and orchestra or between synthesis and control. Aleatora is built around the unifying abstraction of *streams*, which support sequential, parallel, and function composition. Streams span the abstraction hierarchy from patterns all the way down to samples, enabling the composer to compose at any level. The generative composition itself is a first-class value within the language.

The challenge of distribution is addressed by Alternator (Clester & Freeman, 2022), a distribution system that bundles computational compositions (written in existing languages such as Csound, Pure Data, ChucK, Python, JavaScript, C, and Rust, among others) into self-contained packages for client-side execution via WebAssembly. This presents the listener with a familiar music player interface which supports common functions such as pausing and seeking, without requiring additional work on the part

of the composer/programmer. In addition, it enables each listener to hear different pieces and generative variations without requiring a central server to run separate sandboxes for all concurrent listeners.

Finally, the question of environment is taken up by LambDAW (Clester & Freeman, 2023). LambDAW takes the digital audio workstation (DAW) as its starting point and brings computation into the timeline, introducing the concept of *expression items* that generate their contents by evaluating a Python expression. Like spreadsheet formula, these expressions can reference other items in the timeline and transform them, enabling the composer to freely mix code and data, combining the expressive power of programming with the direct manipulation of the DAW. By taking an existing DAW (REAPER) as its starting point, LambDAW augments the DAW's capabilities and offers an entry point into computation for composers and producers who already use the DAW in their creative workflows.

The full set of related work for these three projects is beyond the scope of this short submission; for a more complete overview, see the related work section of each project's paper. To briefly summarize, Aleatora is related to computer music languages such as SuperCollider (McCartney, 2002) and Nyquist (Dannenberg, 1997) and work on streams in general-purpose languages such as Scheme and Haskell. Alternator is related to artistic works such as Generative.fm (Bainter, 2019) and generative radio stations such as `rand()%`[1] and Streaaam (Hollerweger, 2021). And LambDAW is related to environments such as EarSketch (Magerko et al., 2016), Manhattan (Nash, 2014), and DeadCode (Beverley, 2020), as well as work on end-user programming more broadly.

3. Questions for the Community

3.1. Research Questions & Methodology

A key question for the community is how best to pose overarching research questions that relate to this agenda. Of the various frames offered by relevant disciplines (music, creative coding, HCI, Psychology of Programming), which provide the best fit for this kind of work? A related question is what research methodologies may be most effective in answering these questions. Thus far, evaluation has consisted largely of using these systems myself, demonstrating them to others, and discussing their design. What quantitative or qualitative methods (such as surveys, user studies, interviews, etc.) will aid in more rigorously evaluating them? To what extent is practice-led research appropriate for this area of inquiry?

3.2. Potential Directions

There are several potential directions to proceed from my current stage in my research, and I would appreciate community feedback on the merits of these directions and relevant related work. One possibility is "completing the loop." Aleatora already connects nicely to LambDAW, as it is a Python framework well-suited to writing concise expressions that generate audio or MIDI. Completing the loop by building a way to export generative bundles from LambDAW that can run in Alternator would demonstrate a complete approach to generative music, from composition to production to distribution.

Another direction is to enhance Aleatora ('everything as a stream') or Alternator ('everything in a box') by building out infrastructure for embedding more within them, such as SuperCollider and other systems which depend on it. This direction is particularly relevant for recording and replaying live coding performances *as code* (while preserving random elements of the code, which can be re-decided on playback).

A third direction, and perhaps the one most relevant to the PPIG community, is using LambDAW as a jumping-off point to explore hybrid compositional interfaces and programming environments, with the general aim of supporting bricolage by allowing composers to freely mix code and data, taking inspiration from work on interactive visual syntax (Andersen, Ballantyne, & Felleisen, 2020) and *livelits* (Omar et al., 2021). This direction further invites an exploration of how this work—which I have thus far discussed in terms of 'composition' and 'distribution'—relates to liveness, improvisation, and performance. Such an exploration connects to broader questions about the relationship between generative music and live coding (Blackwell & Collins, 2005) and may involve expanding the music/code composition environment beyond conventional computing interfaces such as screens, mice, and keyboards.

[1] `https://www.bbc.co.uk/radio3/cutandsplice/rand.shtml`

4. References

Andersen, L., Ballantyne, M., & Felleisen, M. (2020). Adding Interactive Visual Syntax to Textual Code. *Proc. ACM Program. Lang.*, *4*(OOPSLA). doi: 10.1145/3428290

Bainter, A. (2019). Generative.fm. In A. Xambó, S. R. Martín, & G. Roma (Eds.), *Proceedings of the International Web Audio Conference* (p. 148). Trondheim, Norway: NTNU.

Beverley, J. (2020). Liveness, Code, and DeadCode in Code Jockeying Practice. In *Proceedings of the 2020 International Conference on Live Coding (ICLC2020)* (p. 117-131). Limerick, Ireland: University of Limerick. doi: 10.5281/zenodo.3939222

Blackwell, A. F., & Collins, N. (2005). The Programming Language as a Musical Instrument. In *Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2005)* (p. 120-130).

Cage, J. (2004). Composition as process: indeterminacy. *Christoph Cox, Daniel Warner, Audio Culture: Readings in Modern Music*, 176–187.

Clester, I., & Freeman, J. (2021). Composing the Network with Streams. In *Audio Mostly 2021* (p. 196–199). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3478384.3478416

Clester, I., & Freeman, J. (2022). Alternator: A General-Purpose Generative Music Player. In *Proceedings of the international web audio conference.* Cannes, France: UCA.

Clester, I., & Freeman, J. (2023). LambDAW: Towards a Generative Audio Workstation. In *Proceedings of the 7th International Conference on Live Coding (ICLC2023).* Utrecht, Netherlands. doi: 10.5281/zenodo.7842002

Dannenberg, R. B. (1997). Machine Tongues XIX: Nyquist, a Language for Composition and Sound Synthesis. *Computer Music Journal*, *21*(3), 50–60. (Publisher: The MIT Press) doi: 10.2307/3681013

Eno, B. (1996). *A Year with Swollen Appendices: Brian Eno's Diary*. Faber and Faber.

Essl, K. (2007). Algorithmic composition. In N. Collins & J. d'Escrivan (Eds.), *The cambridge companion to electronic music* (p. 107–125). Cambridge University Press. doi: 10.1017/CCOL9780521868617.008

Hollerweger, F. (2021). Streaaam: A Fully Automated Experimental Audio Streaming Server. In *Audio mostly 2021* (p. 161–168). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3478384.3478426

Magerko, B., Freeman, J., Mcklin, T., Reilly, M., Livingston, E., Mccoid, S., & Crews-Brown, A. (2016). EarSketch: A STEAM-Based Approach for Underrepresented Populations in High School Computer Science Education. *ACM Trans. Comput. Educ.*, *16*(4). doi: 10.1145/2886418

McCartney, J. (2002). Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal*, *26*, 61–68. doi: 10.1162/014892602320991383

Nash, C. (2014). Manhattan: End-User Programming for Music. In *Proceedings of the international conference on new interfaces for musical expression* (pp. 221–226). London, United Kingdom: Goldsmiths, University of London. doi: 10.5281/zenodo.1178891

Omar, C., Moon, D., Blinn, A., Voysey, I., Collins, N., & Chugh, R. (2021). Filling Typed Holes with Live GUIs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (p. 511–525). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3453483.3454059

Reich, S. (1965). Music as a gradual process. *Writings on music*, *2000*, 34–36.