Composing with Generative Systems in the Digital Audio Workstation

Ian Clester, Jason Freeman

Georgia Institute of Technology, Atlanta, GA, USA

Abstract

Generative systems present new opportunities for composers, but it can be unclear how to integrate such systems into creative workflows. We put forward a vision for a *generative audio workstation*, in which the composer can work with generative expressions much like ordinary audio or MIDI items, seamlessly mixing static and dynamic musical material. We present our research prototype in this direction, LambDAW, which takes the form of an extension to REAPER (a popular digital audio workstation) that executes Python expressions directly in the timeline, and we discuss possibilities for integration with generative models and machine learning libraries.

Keywords

generative music, end-user programming, music composition, digital audio workstation

1. Vision

Generative musical systems have a long history [3], from Hiller and Isaacson's *Illiac Suite* in 1957, to David Cope's *Experiments in Musical Intelligence* in the 1980s [4], to deep neural nets such as OpenAI's *MuseNet* or Magenta's *Music Transformer* today. In all these cases, the computer is entirely relied on to generate the final musical material. Human input ends after constructing the program and possibly providing a prompt.

This workflow contrasts with the more hands-on approach enabled by the digital audio workstation (DAW), in which the composer can directly manipulate and arrange items along a timeline, making it easy to import and work with disparate materials. However, this approach affords limited support for generative music. To incorporate generative systems into a piece, the composer has two options.[1] They can abandon the DAW in favor of a computer music environment (e.g. Csound, SuperCollider, Max) or general-purpose language that allows programming the system directly. Or they can run the generative system separately, generate some audio or MIDI output, and then import that into the DAW, going back and forth each time they want to tweak the system or generate a different output.

We put forward a vision for a *generative audio workstation*: an environment as adept with generative audio as DAWs are with (static) digital audio. In our vision, code is not something separate to be executed outside of the DAW, nor is it timeless in the FX chain. Instead, it is right in the timeline, alongside the other musical materials. Furthermore, it can connect to other materials by reference, enabling the composer to create meaningful links through the piece. In this model, the composer need not give up all control to generative systems, nor reject them entirely. Instead, they retain ultimate control of the piece and can bring in generative systems as they see fit, like any other source of material.

We take inspiration from end-user programming software, including the classic example of spreadsheets and more recent work from Ink & Switch [5, 6]. We also draw on computational notebooks such as Jupyter [7], which promote interactive programming and allow code to generate pieces of the document it is embedded in. Our work is also related to *Manhattan* [8] (which brings code fragments into a music tracker), *Ossia Score* [9] (an "intermedia sequencer" which can be scripted via JavaScript), and *Computational Counterpoint Marks* [10] (which brings code into Western music notation).

2. Prototype

Our research prototype takes the form of an extension to REAPER called LambDAW ("lambda" + "DAW").[2] With LambDAW, the composer can write Python expressions in the timeline to generate audio or MIDI output directly in the DAW, as shown in Fig. 1. These expressions are stored in take names, so that both the code and its generated output are visible in the timeline. Like a spreadsheet formula, a name begins with = to indicate that it contains

Joint Proceedings of the ACM IUI Workshops 2023, March 2023, Sydney, Australia

ijc@gatech.edu (I. Clester); jason.freeman@gatech.edu (J. Freeman)

https://ijc8.me/ (I. Clester); https://distributedmusic.gatech.edu/ (J. Freeman)

[1] They can also put the generative system in a plugin, but this is hidden in the FX chain, exists for all time, and has limited ability to interact with the DAW.

[2] LambDAW is free software, and it is available at https://github.com/ijc8/lambdaw.

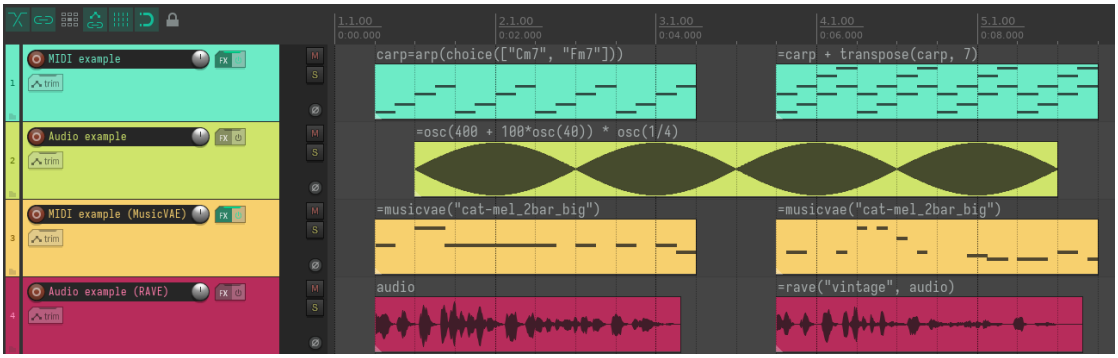

Figure 1: LambDAW allows the user to embed Python expressions that generate audio or MIDI directly in the DAW timeline. In this screenshot, tracks 1 & 2 show examples of expression items that programmatically generate audio and MIDI, while tracks 3 & 4 show examples that invoke ML models (MusicVAE [1] for melody generation and RAVE [2] for timbre transfer).

an expression to be evaluated. (The `=` may be further prefixed to give it a name for later reference, as in `foo=bar()`.) When LambDAW detects a new or updated expression, it automatically evaluates it and puts the generated output in the associated item. The user can also re-evaluate expressions on demand, e.g. to get different outputs from a model.

Expressions can refer to other items in the timeline as variables. For example, if there is a MIDI item in the timeline named `my_cool_riff`, an expression item can refer to it with an expression like `=transpose(my_cool_riff, 5)`. This feature allows the composer to establish connections between musical material; if they later modify the original riff, the derived item can be updated simply be re-evaluating it (unlike a transformed copy, which "forgets" its relationship to the original material). Referring to items in expressions also facilitates the transformation of recorded material with code. (Expression items can also reference other expression items, as in the example with `carp` in track 1 of Fig. 1.)

Linking expressions by reference enables the user to divide up complexity between expressions and so provides a way to manage complexity and expression length; the *user project module* provides another.[3] LambDAW loads a user-defined module for each project in which the user can define functions, import useful libraries, load resources, etc., for use in timeline-embedded expressions. The user can also customize how LambDAW converts DAW items to/from Python objects. The user project module thus supports concision and customizability, enabling the user to choose their own set of abstractions for composing and maintain the brevity of expressions in the timeline.

[3] An expression in the timeline can be arbitrarily complex, but it is advisable to keep it brief so that the whole thing can be seen at a glance and without excessive zooming.

3. Conclusion & Future Work

We believe LambDAW offers a useful way to compose with AI and ML by bringing code into the familiar interface of the DAW. Because LambDAW allows arbitrary Python expressions in the timeline and provides the project module as a place to import libraries and load assets, it enables the user to take advantage of Python's rich ecosystem. Expression items can serve as user-specified "blanks" for an AI to fill in, which the user can re-evaluate to generate new output. The ability of LambDAW expressions to refer to other items in the timeline provides a convenient way to work with models that transform or continue input material, and for the user to pre- or post-process model data.

Looking ahead, we plan to further explore integrations with generative libraries and models to find how they can fit into compositional workflows. Some initial work in this direction is depicted in Fig. 1, which features expressions that invoke MusicVAE [1] and RAVE [2] to generate symbolic data and audio, respectively. We also aim to make architectural changes to improve LambDAW's usability (especially with ML libraries), such as performing evaluation in a separate process and allowing to user to interrupt long-running operations.[4] Finally, we hope to present our system to composers to get their feedback about the quality of the integration, the degree to which it facilitates their use of generative systems, and how it affects their creative process.

[4] While building our prototype, we encountered technical issues with REAPER's embedded Python support when using libraries such as NumPy, TensorFlow, and PyTorch, which required workarounds. Moving evaluation out to a separate process (as in Jupyter kernels) would avoid these issues, enable interrupting/killing the interpreter without restarting the DAW, and prevent evaluation from blocking the UI thread.

References

[1] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, D. Eck, A hierarchical latent vector model for learning long-term structure in music, in: International Conference on Machine Learning (ICML), 2018. URL: http://proceedings.mlr.press/v80/roberts18a.html.

[2] A. Caillon, P. Esling, Rave: A variational autoencoder for fast and high-quality neural audio synthesis, 2021. URL: https://arxiv.org/abs/2111.05011. doi:10.48550/ARXIV.2111.05011.

[3] K. Essl, Algorithmic composition, in: N. Collins, J. d'Escrivan (Eds.), The Cambridge Companion to Electronic Music, Cambridge Companions to Music, Cambridge University Press, 2007, p. 107–125. doi:10.1017/CCOL9780521868617.008.

[4] D. Cope, Experiments in musical intelligence (emi): Non-linear linguistic-based composition, Interface 18 (1989) 117–139. URL: https://doi.org/10.1080/09298218908570541. doi:10.1080/09298218908570541.

[5] G. Litt, M. Schoening, P. Shen, P. Sonnentag, Potluck: Dynamic documents as personal software (2022). URL: https://www.inkandswitch.com/potluck/.

[6] J. Lindenbaum, S. Kaliski, J. Horowitz, Inkbase: Programmable ink (2022). URL: https://www.inkandswitch.com/inkbase/.

[7] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, J. development team, Jupyter notebooks - a publishing format for reproducible computational workflows, in: F. Loizides, B. Scmidt (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas, IOS Press, Netherlands, 2016, pp. 87–90. URL: https://eprints.soton.ac.uk/403913/.

[8] C. Nash, Manhattan: End-user programming for music, in: Proceedings of the 14th International Conference on New Interfaces for Musical Expression, 2014, pp. 221–226. URL: https://www.nime.org/proceedings/2014/nime2014_371.pdf.

[9] J.-M. Celerier, P. Baltazar, C. Bossut, N. Vuaille, J.-M. Couturier, M. Desainte-Catherine, Ossia: Towards a unified interface for scoring time and interaction, in: TENOR 2015 - First International Conference on Technologies for Music Notation and Representation, 2015. URL: http://tenor2015.tenor-conference.org/papers/13-Celerier-OSSIA.pdf.

[10] J. C. Martinez, Extending music notation as a programming language for interactive music, in: ACM International Conference on Interactive Media Experiences, ACM, 2021, pp. 28–36. URL: https://doi.org/10.1145/3452918.3458807.