Lessons from Building a Multi-Language Live Coding Environment for the Web

How schema-first design enabled multi-language support in Minuet

Jason Gwartz

Introduction

Minuet is a web-based live-coding environment that supports multiple programming languages through a schema-first architecture. By decoupling the performer's programming language from the audio engine, Minuet can scale from gentle introduction of programming fundamentals to maximally-expressive code for live performance. Minuet is built entirely in the browser, based on Web Audio and Web MIDI APIs.
This talk will discuss the motivations and tradeoffs involved in this architecture, and implementation challenges experienced during development. A demonstration of Minuet will involve live-coding in the browser in multiple programming languages (TypeScript, Python, and others), as well as Minuet's integration with hardware MIDI controllers and analog synthesisers.

Deployment: https://minuet.gwartz.me/ Source code: https://github.com/jasongwartz/minuet

Description

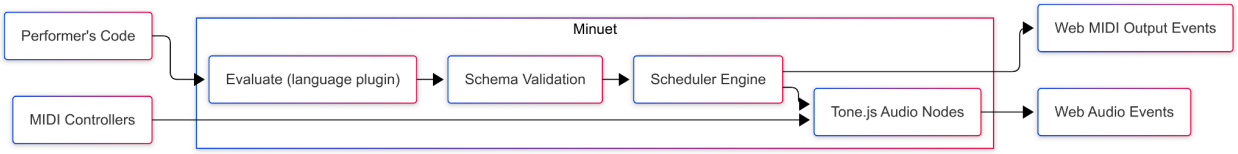
Minuet is a web-based system for loop-based live-coding music with support for samples, software synthesizers, effects chains, and external MIDI devices like hardware controllers and analog synthesizers. Minuet brings all of these features to any programming language that can be evaluated in the browser by JavaScript or WebAssembly (WASM).

Other live-coding systems bring some but not all of the features or accessibility of Minuet. Sonic Pi[1][2] uses Ruby (declining in popularity for new developers[3]) and procedural timing (play-sleep-play). Tidal Cycles[4] and the web port Strudel[5] use Haskell as the performer language, and Extempore[6] uses Scheme, making them complex for a pedagogical tool. Gibber[7] runs in-browser and has excellent documentation, but lacks first-class support for MIDI controllers or output.

Minuet's architecture is based on the principle that musical intent can be separated from programming language syntax. The musical meaning of `` `{ "on": 2, sample: "kick" }` `` is identical to `` `sleep 2; play("kick")` ``, both being expressions of the same musical event. In Minuet, the code

[1] https://sonic-pi.net/
[2] "Using the Sonic Pi Application for Educational Purposes – A Literature Review", Marius Bănuț, 2023.
[3] Stack Overflow Developer Survey 2024
[4] https://tidalcycles.org/
[5] https://strudel.cc/
[6] https://github.com/digego/extempore
[7] https://gibber.cc/

the performer writes does not directly schedule audio events; it builds a declarative data structure with the desired scheduling plan. This data structure is validated against Minuet's schema, followed by the scheduling of the resulting events by the Tone.js-based audio engine.



This separation of interface from implementation mirrors patterns from distributed systems, where teams collaborate by writing disparate services with APIs[8]. In Minuet, this decoupling means each supported language only needs to be able to output the data structure, not implement the entire scheduling system. Minuet currently has built-in support for TypeScript, Python, Lua, Jsonnet, and YAML; it can be extended to support any browser--executable language.

Minuet is built with a modern frontend stack; React is by far the most widely-adopted frontend framework[9][10], but requires careful state management for stable long-running live performances. Minuet initially used React props to pass pieces of state to reactive components (like the "current beat" indicator and the audio meters). This was frequently causing thousands of unnecessary re-renders, which would lead to laggy audio processing or the whole application crashing. Component memoization and stateful "atoms" for cross-component dependencies help avoid unnecessary component recreation.

Adding support for MIDI controllers and synthesizers presented a synchronisation challenge[11], since the Web MIDI API does not have access to the Web Audio clock and it is not possible to schedule Web MIDI events directly in Web Audio time[12]. Minuet schedules each MIDI note event as a Tone.js Draw() event, which can schedule a callback on the Web Audio clock with high precision[13].

Minuet attempts a "graceful degradation" approach to handle programmer errors without musical interruption. If an individual track has a scheduling error (like an invalid time denotation), that track will be ignored during scheduling but other tracks will still be played. If the entire schema is deemed invalid, Minuet will skip scheduling of the new configuration, and instead proceed with a repeat loop of the last valid configuration.

In conclusion, building a complete web-based live-coding environment required solving fundamental web-specific challenges with state management, clock synchronization, and graceful error handling. The schema-first architecture enabled Minuet to become a multi-language programming environment with the unique portability of the web as a platform.

---

[8] The Bezos Mandate, as told by Steve Yegge: https://gist.github.com/chitchcock/1281611
[9] Stack Overflow Developer Survey 2024
[10] The State of Frontend Developer Survey 2024
[11] A Tale of Two Clocks, Chris Wilson
[12] GitHub issues: https://github.com/WebAudio/web-midi-api/issues/180, https://github.com/WebAudio/web-midi-api/issues/232
[13] https://github.com/Tonejs/Tone.js/wiki/Performance#syncing-visuals

Biography

Jason Gwartz is a software developer, tech leader, and jazz musician. He currently works at the UK AI Security Institute as Head of Core Technology, and has previously led tech teams at climate-tech startups and worked at Apple on iCloud privacy. He also performs regularly on the saxophone with London's Duke Street Big Band and the folk-punk group The Great Malarkey. Jason is the author of Chopsticks, an educational GUI live-coding system previously presented at WAC 2018. He holds an MSc in Computer Science from University College London, and a BFA in Music from York University.