Alternator: A Computational Music Player

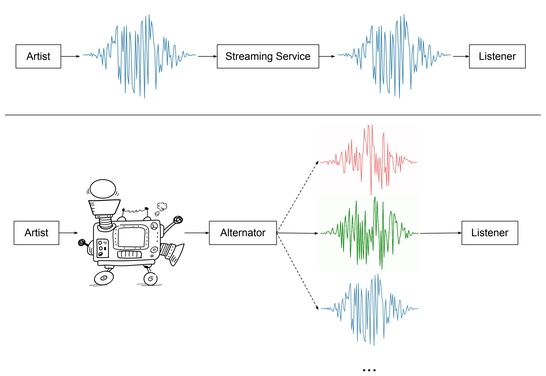
Ian Clester (ijc@gatech.edu)

Georgia Tech Center for Music Technology
College of Architecture

INTRODUCTION

Artists typically release music by creating an **audio recording**. Digital recordings can be **easily distributed** to many listeners, who can then listen to the music whenever and however they want. As an audio recording, the music will be **exactly the same** each time is played — unlike live performance.

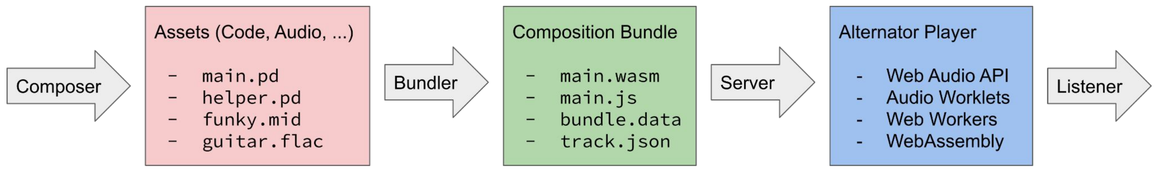
In this sense, **recording is a tradeoff**: it sacrifices **dynamicity and variation** for **convenience and reproducibility**. **Alternator** avoids this tradeoff by distributing music as a **recorded computation** rather than **recorded audio**. In other words, the artist creates a machine that generates music (a program), and the listener hears its output, which **may change every time it is played**.



Conventional streaming model (top) vs. Alternator (bottom)

ARCHITECTURE

Alternator works by executing **bundles** containing everything necessary to run a piece: code (e.g. Pure Data patches, Csound scores) and data (audio samples, MIDI, trained models). The composer uses a tool provided with Alternator to create a bundle, choosing a **template** based on **their language of choice**.
The Alternator client is implemented as a web app; it can run on any device with a modern browser. It executes bundles using **WebAssembly**, with common language runtimes compiled using **Emscripten**. By executing bundles on the client-side (the user's device) rather than server-side, Alternator can scale up readily without requiring excessive server resources.

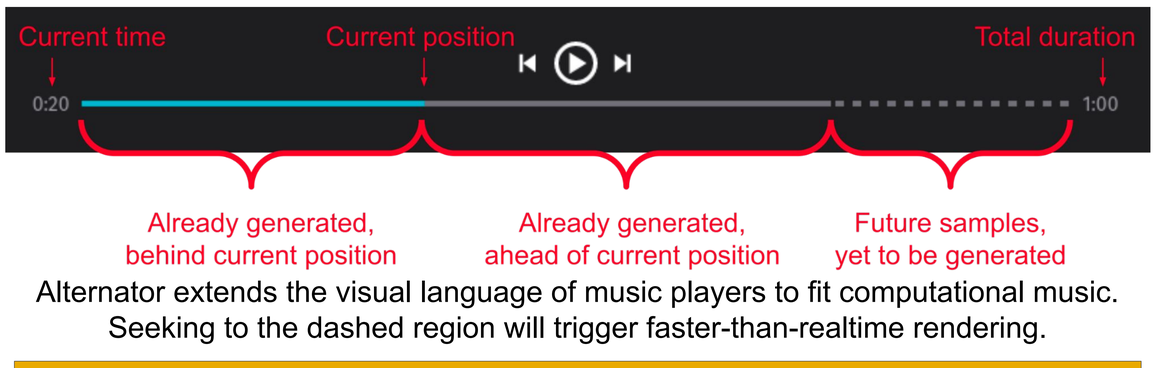


GENERALITY

Alternator does not have special support for particular audio languages built-in. Instead, it provides a **general mechanism** to load and execute bundles containing JavaScript, WebAssembly, and other assets. Ultimately, a piece just needs some way to **fill a buffer with samples** in order to run in Alternator.
For convenience, **templates** are provided for common audio languages such as **Pure Data**, **Csound**, and **RTcmix**, as well as general-purpose languages such as **Python** (e.g. with Aleatora!), **JavaScript**, **C/C++**, and **Rust**.

USER INTERFACE

Alternator adopts a **familiar interface**; it **translates** common features of a conventional music player into a **compositional context**. This is straightforward for some features, but for others it requires more consideration. For example, **what does it mean to seek** when future audio does not yet exist? In Alternator, this renders **faster-than-realtime** to the target time. Similarly, seeking backwards goes into a **history buffer** specific to each playthrough. Even displaying track durations requires some consideration when **tracks may be infinite** or have a **range of possible durations**. Alternator aims to gracefully handle these cases in an otherwise-conventional interface. The basic metaphors still apply (control over something that autonomously generates sound), so Alternator **extends** rather than replaces **the interface listeners already know**.



Alternator extends the visual language of music players to fit computational music. Seeking to the dashed region will trigger faster-than-realtime rendering.

BACKEND

Because Alternator **generates audio on the client-side**, it does not ask much of the backend. For a proof-of-concept, static hosting (for bundles) is sufficient; for features such as search, user playlists, and recommendations, a more full-featured backend is required. For now, Alternator (inspired by utteranc.es) uses GitHub for **static hosting** and **search**. A composer can get their work in Alternator by creating a **repository of bundled tracks** with the `#alternator-album` tag.

DISCUSSION AND NEXT STEPS

Distributing **music as code** opens up new possibilities. Alternator includes a "**view source**" interface, allowing the listener to see the contents of the bundle and a link to the repository. This invites the listener to **see how the music is put together**, and even (for a sufficiently intrigued listener) try tweaking or forking it.
Alternator focuses on **purely generative music** for interface consistency, but there is room to **explore** how this model could work for **interactive music** in the future.

TRY IT OUT

Try Alternator out at ijc8.me/alternator!
Find it on GitHub at https://github.com/ijc8/alternator.