Flow - A framework for *browser-based interactive* audio applications

Andrew Thompson & Gyorgy Fazekas

Keywords: Web Audio API / declarative programming / Model-View-Update

The Web Audio API brings real time audio synthesis and processing to the browser with a high-level JavaScript API. A number of frameworks have since been developed to make Web Audio application development easier such as BRAID [1], WAAX [2], and Flocking [3].

When reviewing these libraries we have identified **two problems** in how these frameworks suggest programs should be structured:

1) Audio and UI Coupling: Audio nodes are manipulated directly in UI event callbacks. This coupling makes it ***diffficult to change*** one without impacting the other.

2) Fragmented State: Audio and UI elements are *stateful objects*. This causes application state to become scattered around the codebase, making it ***difficult to track changes*** and find bugs.

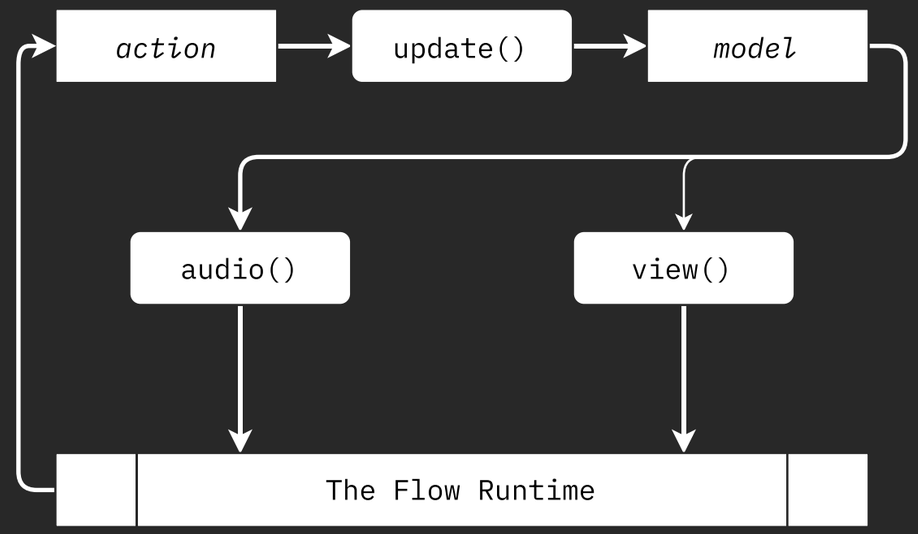
Flow takes a different approach by mandating that all audio and UI elements must be created from a ***single model of application state***, and neither can manipulate that state directly.

Scan the QR code with your mobile's camera to access the full paper.

Additionally Flow's documentation can be found by heading to: **b.link/flow-docs**

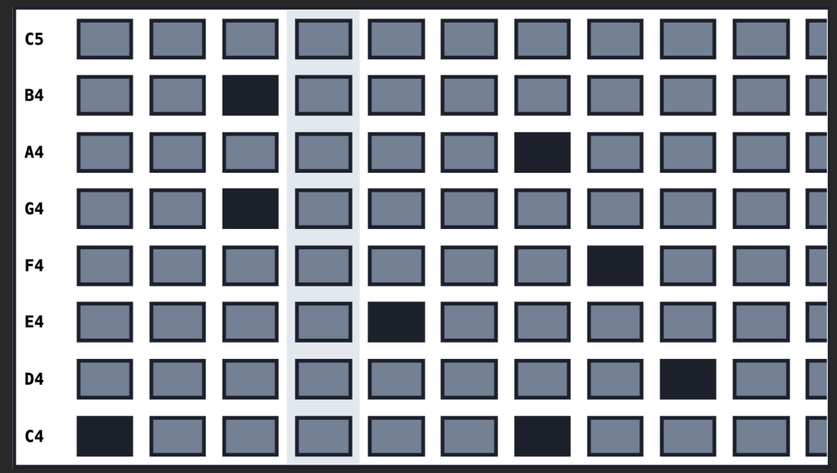
[1] Ben Taylor and Jesse Allison. 2015. BRAID: A web audio instrument builder with embedded code blocks. In Proceedings of the 1st interna- tional Web Audio Conference.
[2] Hongchan Choi and Jonathan Berger. 2013. WAAX: Web Audio API eXtension.. In NIME. 499–502.
[3] Colin BD Clark and Adam Tindale. 2014. Flocking: a framework for declarative music-making on the Web. In SMC Conference and Summer School. 1550–1557.

EPSRC+AHRC Centre for Doctoral Training in Media and Arts Technology EP/L01632X/1



Flow enforces the **Model-View-Update** architecture to allow for clear, deterministic updates to application state. **Actions** are dispatched by the runtime in response to some event. A single **update** function then determines how to create a new application **model**. This single model is then used by the **audio** and **view** functions to generate the audio graph and HTML output respectively.

```
function audio (model) {
  return model.steps.map(({ note, active }) ⇒
    N.oscillator([ P.frequency(note) ], [
      N.gain( [ P.gain(active ? 1 : 0) ], [
        N.dac()
      ])
    ])
  )
}
```



Arts & Humanities Research Council
EPSRC Engineering and Physical Sciences Research Council
Queen Mary University of London