

Notes

10 min presentation, 10 min demo
see more in presentation notes below

Real-Time Interactive Music In Haskell

Paul Hudak, Donya Quick,
Mark Santolucito, Daniel Winograd-Cort



CREATIVE CONSILIENCE
of Computing and the Arts

Functional Reactive Programming

- Used for time-varying systems
- Abstraction of time fits music well
 - Simple and concise code
 - Works well in a classroom setting
 - Rapid prototyping

Euterpea

- Arrowized-FRP EDSL for computer music
- Written in Haskell
- Provides:
 - Realtime MIDI production
 - Built-in MIDI I/O
 - Waveform manipulation and synthesis

The Haskell School of Music

— From Signals to Symphonies —



Paul Hudak

**Yale University
Department of Computer Science**



euterpea.com

Euterpea Code About Community Install

Learn

There are lots of resources to learn Euterpea. The most developed is the textbook, Haskell School of Music by Paul Hudak. It is freely available online from the [Yale Haskell Group](#). Every year a course is taught at Yale about Computer Music using Euterpea. Homework assignments can be viewed on the [course website](#). For more official documentation you can check the [Hackage page](#), or view the latest version on [github](#).

We are also developing some tutorials so you can learn by example here. In addition to some bite size walkthroughs, we will also be posting user submitted code. If you make something you would like us to show off, just email Mark the source file!

Code

- [midi/basics.lhs](#)
- [midi/Lecture6.lhs](#)
- [midi/chorale.lhs](#)
- [midi/scales.lhs](#)
- [midi/rhythm.lhs](#)
- [examples/dub.lhs](#)
- [examples/BabaORiley.lhs](#)

Real-time *Interactive* Music

- Tidal, live coding in ghci
 - Supercollider, imperative commands
 - Csound, written in C
-
- We would like this for FRP (in Haskell)

Real-time *Interactive* Music

- Euterpea has poor support for this.
- Naively connecting a GUI can lead to:
 - High audio latency
 - Unpredictable performance
- Also, ad-hoc inter-library connections are:
 - Not extensible
 - Difficult to maintain

Media Modules

- Different media types on different threads
 - No data rate bottlenecks, even with varying rates
- Easy interlibrary communication
 - Abstraction agnostic

Media Modules

- We made an attempt at a universal FRP API:

Media Modules

- We made an attempt at a universal FRP API:
 - A generic Arrow type that supports IO

```
class Arrow a => ArrowIO a where  
  liftAIO :: (b -> IO c) -> a b c  
  
type IOAuto = Automaton (Kleisli IO)
```

Media Modules

- We made an attempt at a universal FRP API:
 - A generic Arrow type that supports IO
 - Asynchronous inter-library operators

```
asyncCIO :: (ArrowIO a, NFData c) =>
    (IO d, d -> IO ())
  -> (d -> IOAuto b c)
  -> a b [c]
```

Media Modules

- We made an attempt at a universal FRP API:
 - A generic Arrow type that supports IO
 - Asynchronous inter-operative operators
- We built these concepts into the FRP GUI library UISF.

Connecting UISF and Euterpea

- Euterpea builds a UISF widget for midiOut:

```
midiOut :: ArrowIO a =>
  a (OutputDeviceID, [MidiMessage]) ()

midiOut = liftAIO action where
  action (dev, mm) = do
    outputMidi dev
    forM_ mm (\m -> deliverMidi dev (0, m))
```

Connecting UISF and Euterpea

- Now we lift that asynchronously to UISF:

```
asyncMidi :: NFData c =>
  PureAuto b ((OutputDeviceID, [MidiMessage]), Int, c)
  -> UISF b [c]
asyncMidi sf = asyncCIO (return (), const $ return ()) sf
  where sf = proc b -> do
    (mdata, t, c) <- liftAutoIO sf -< b
    midiOut -< mdata
    liftAIO threadDelay -< t
    returnA -< c
```

Demo: UISF + Euterpea

- Media module design allows seamless operation.
 - The system overcomes performance issues.
 - Underlying design remains pure and simple.
- Our demo stress tests the ideas.
 - Multi-part UI
 - Hard music from Kullita

Demo: Elerea + Euterpea

- We took a similar approach with Elerea.
 - It's currently less polished,
 - But we get good performance and clean code.

Future Work

- Formalize the media module API.
 - How can we make different media libraries inter-operate easily and efficiently?
- Retrofit more libraries into media modules.
 - Extend the inter-operability to other media systems.
 - Talk to us about how we can incorporate your system!

Conclusions

- Euterpea and UISF work together easily.
 - They retain a relatively pure, functional style.
 - A great tool for teaching functional computer music.
- We encourage users to play with the system:
 - `euterpea.com`
 - `haskell.cs.yale.edu`
 - `cabal install euterpea`
 - `cabal install uisf`

