Real-Time Interactive Music in Haskell

Paul Hudak Donya Quick Mark Santolucito Daniel Winograd-Cort
Yale University Department of Computer Science, USA
{paul.hudak, donya.quick, mark.santolucito, daniel.winograd-cort}@yale.edu

Abstract

Euterpea and UISF are two recently released Haskell libraries on Hackage that facilitate the creation of interactive musical programs. We show an example of using these two libraries in combination with Haskell's support for parallelism to create a complex application that generates music in real time in response to user input from MIDI controllers.

Categories and Subject Descriptors H.5.5 [*Information Interfaces And Presentation*]: Information Systems; D.1.1 [*Programming Techniques*]: Applicative (Functional) Programming

Keywords Haskell, Euterpea, UISF, arrows, functional reactive programming

1. Introduction

We present a Haskell-based music generation application that illustrates the use of two libraries, Euterpea and UISF, for creating interactive musical applications. The program has a graphical interface that allows the user to select different properties about the generated music. It also takes input from MIDI controllers, the pitches of which are used to determine the key, bassline, and harmony of the generated music. Although currently limited to producing a single style of music, this program demonstrates the potential of using the Euterpea and UISF libraries in Haskell for building large-scale interactive musical systems. These libraries are also useful in a classroom setting, providing a user-friendly way for students to build interactive programs with relatively little code.

This combination of libraries is a unique example of support for interactive multimedia systems in a pure functional style. Supercollider is a popular language for creating interactive music[5], but uses a very impure functional style. Python is another functional language that has support for creating interactive musical systems with libraries like JythonMusic[4], but applications using these libraries tend to be coded in a more imperative style. In pure functional language development, most work addresses interactive media of a single type. For example, Tidal is a Haskell library for live coding that supports musical applications[6]. Since the fundamental interface in live coding is code itself, interactive multimedia is not required in the same way it is presented here. In contrast to these other libraries and coding environments, the combination of Euterpea and UISF allows the creation of multimedia systems while retaining the benefits of coding in a pure functional style with Haskell: concise code, rapid prototyping, and a rich type system that eliminates many categories of bugs experienced by imperative programmers.

2. Euterpea

Euterpea[2] is a domain-specific language for computer music development embedded in the functional language Haskell[7]. In addition to data structures for representing many features of music and various algorithms for offline music generation, Euterpea also features support for real-time MIDI I/O, allowing for the creation of interactive musical applications.

Euterpea's real-time MIDI I/O interface wraps the low-level implementations of two other Music-related Haskell libraries, HCodecs[1] and PortMidi[3], and provides a simpler interface to the user that allows pure handling of MIDI events. The performance of systems written with this approach is good enough that even complex programs can be very responsive and low-latency. Using features of the UISF library, Euterpea can also be used to build musical user interfaces, or MUIs.

3. The UISF Library

UISF[10] is a functional reactive Haskell library for creating graphical user interfaces. Its design is based on the theory of *arrows*, and thus its standard use employs Haskell's arrow syntax. In practice, this means that adding widgets to a UISF GUI involves adding lines of code to one's program that look as follows:

```
outputStream <- widget -< inputStream
```

Notice that the special characters (`<-` ... `-<`) create the shape of an arrow pointing to the left: this means that the input stream is processed through the widget into an output stream.

The library supports standard widgets such as labels, buttons, sliders, text boxes, check boxes, etc, and the GUI layout can be changed with transformer functions such as `leftRight` and `topDown`.

Euterpea contains additional widgets to lift real-time MIDI I/O operations into UISF's arrowized environment for creating MUIs. Two such widgets are *midiIn* and *midiOut*, which allow the user to receive and send MIDI messages respectively within a UISF context. A common use case for these widgets is to take MIDI input from some device, perform computation on that input, and then send a collection of new messages to another device. The general format of code for this is quite straightforward:

```
midiMsgs <- midiIn -< inputDevice
let newMsgs = f midiMsgs
midiOut -< (outputDevice, newMsgs)
```

The simplicity of these widgets makes them an excellent candidate for simple MIDI applications, such as music-related home-

FARM'15, September 5, 2015, Vancouver, BC, Canada
ACM. 978-1-4503-3806-6/15/09
http://dx.doi.org/10.1145/2808083.2808087

work assignments and term projects for students learning about Haskell. However, these widgets suffer from one problem: the handling of MIDI events in the code above is actually tied to the update rate of the UISF-based graphical interface—which is in turn limited by properties of the underlying graphics implementation. An update rate of 60 frames per second is sufficient for graphics, but is slow for many types of music. As a result, computation-heavy programs that also require precise timing at fast tempos can experience audible timing issues. Fortunately, there is an easy and user-friendly way to overcome this and achieve good performance even under demanding conditions.

4. Improving Performance with Parallelism

Previously, we presented a technique of *media modules* to create intermedia and multi-rate applications[9]. The media module design allows different media-producing components to communicate without requiring significant structural changes to individual modules or impacting performance.

We incorporated the media module design directly into Euterpea and UISF to create a new parallel, multi-rate, low-latency widget for MIDI output. This new widget takes as an argument a function that produces MIDI messages as well as timing information of how this function should run and automatically parallelizes it and sets up the appropriate communication channels. In practice, the user need only exchange the previously stated *midiOut* widget with a line such as:

```
asyncMidi midiGen -< (outputDevice, midiInput)
```

where *midiGen* is an algorithm for processing the MIDI messages in *midiInput*. A key point of this technique is that any potentially unsafe effects that are typical of asynchrony and shared memory are not directly exposed to the users. Instead, we allow the user to write pure Haskell code to describe the behavior of the individual components (music generation and user interface) without worrying about the underlying implementation.

5. Application: A Three-Part Music Generator

To demonstrate the use of Euterpea and UISF, we present a program that generates music stochastically based on user input[1]. The program uses UISF to allow the user to select properties of the generated music and iterates through the following two-step pattern to generate a series of MIDI messages as output:

1. Over a fixed-length cycle (either a beat or a measure depending on the selected settings), MIDI input is taken from the MIDI devices specified via the GUI and placed in a buffer. The buffer is cleared after each cycle. MIDI input can be taken from any number of MIDI input devices.
2. A key-finding algorithm is run on the pitches received during the last cycle. The determined key is used as input for generating music in the next cycle.

The user is able to alter the key of the harmony and the bassline by providing input from a standard MIDI controller. For example, if the currently playing music is in C-major and the user plays an E-flat-major triad (the pitch classes E-flat, G, and B-flat), the program should switch to E-flat major. Depending on the settings chosen, the bassline will either use primarily the root and fifth of the key or it will be stochastically chosen from the pitch classes in the user input. The harmony at each step is generated stochastically using a Haskell implementation for chord spaces[8], which are a way of mathematically grouping collections of pitches in musically meaningful ways. The overall style of the output produced is similar to pop music, and the chord spaces used generate jazzy harmonies. Using the approach described in section 4, the program's performance is smooth and responsive. Figure 1 shows the overall pattern of communication between MIDI devices and threads in the program.

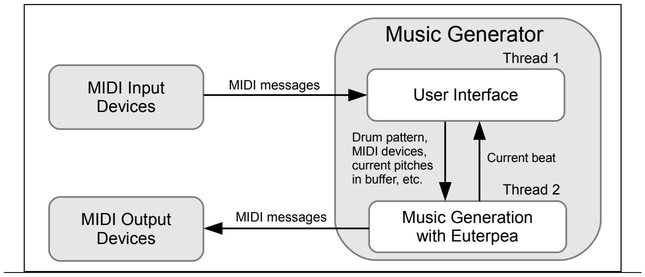

Figure 1. Illustration of information flow between the two threads of the music generator program. Because the timing of the user input is less critical, MIDI input can be taken at the slower UISF update rate without harming performance.

6. Conclusion

Euterpea and UISF are powerful libraries for creating interactive musical programs in Haskell. Euterpea allows easy manipulation of musical structures, and UISF enables the creation of graphical interfaces with a concise and elegant coding style. These two libraries are very easy to use, making them a great tool for teaching functional programming techniques in a musical context.

Acknowledgments

This research was supported in part by NSF grant CCF-0811665. Authors are listed in alphabetical order and contributed equally to the work presented in this article.

References

[1] G. Giorgidze. HCodecs, 2014. URL `https://hackage.haskell.org/package/HCodecs`.

[2] P. Hudak. Euterpea, 2014. URL `http://haskell.cs.yale.edu/euterpea/`.

[3] P. H. Liu. PortMidi, 2015. URL `https://hackage.haskell.org/package/PortMidi`.

[4] B. Manaris, A. R. Brown, and T. Kohn. Making music with computers: Creative programming in python. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 705–705. ACM, 2015.

[5] J. McCartney. Rethinking the computer music language: SuperCollider. *Comput. Music J.*, 26(4):61–68, Dec. 2002. ISSN 0148-9267.

[6] A. McLean. Making programming languages to dance to: live coding with tidal. In *Proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design*, pages 63–70. ACM, 2014.

[7] S. Peyton Jones. The Haskell 98 Language and Libraries: the Revised Report. *Journal of Functional Programming*, 13(1):0–255, Jan 2003.

[8] D. Quick and P. Hudak. Grammar-Based Automated Music Composition in Haskell. In *Proceedings of the first ACM SIGPLAN workshop on Functional Art, Music, Modeling, and Design*, pages 59–70, 2013.

[9] M. Santolucito, D. Quick, and P. Hudak. Media Modules: Intermedia Systems in a Pure Functional Paradigm (forthcoming). In *Proceedings of International Computer Music Conference*, 2015.

[10] D. Winograd-Cort. UISF, 2015. URL `https://hackage.haskell.org/package/UISF`.

[1] Our implementation is available at `http://haskell.cs.yale.edu`.