Kairos - a Haskell Library for Live Coding Csound Performances

Leonardo Foletto

Berklee College of Music
fltleonardo@gmail.com

Abstract. Kairos [1] is a library for the Haskell programming language designed to live code patterns of Csound score instructions to be sent to a running UDP server [2] of a pre prepared Csound orchestra.

Keywords: Live Coding, Csound, Haskell

1 Introduction

Within the context of the arts, live coding has seen an increasing adoption. Today, a growing community of artists who, already accustomed to using software as a tool in their artistic practice, strive to find new methods to interact with their machines in more personal and meaningful ways. In the past ten years, many open source software tools have been developed to perform and compose live coded music including SuperCollider[3], TidalCycles [4], Conductive[5], Sonic Pi[6] .

Kairos is a live coding system inspired by some of the operational principles of the above mentioned softwares, but born from the need of the author to create an environment to perform and compose music more closer to his needs that feels intuitive to use and provides a high degree of control with a simple, yet powerful syntax.

2 System Overview

There are two main parts to the system: a Csound file, `kairos.csd`, and an accompaning Haskell library. The focus of the project has been developing an Haskell library to format patterns of data into Csound readable score to be sent over a UDP network to a running instance of a Csound server.

2.1 Csound File

The file `kairos.csd` contains a number of pre defined instruments and global effects. All the instruments have been programmed to have a positive finite value of `p3` and are written to maximize the number of common pfields amongst all the instruments.

Pfields shared amongst every instrument

```
 p4 : amplitude (0 - 1)
 p5 : reverb send (0 - 1)
 p6 : delay send (0 - 1)
 p7 : panning (0 - 1)
```

Example of a simple sampler instrument

```
<CsInstruments>

instr 1 ;Sampler

inchs filenchnls p8

if inchs = 1 then
aLeft diskin2 p8, p9
outs aLeft*p4* sqrt(1-p7), aLeft*p4* sqrt(p7)
garvbL = garvbL + p5 * aLeft  * sqrt(1-p7)
garvbR = garvbR + p5 * aLeft  * sqrt(p7)
gadelL = gadelL + aLeft   * p6 * sqrt(1-p7)
gadelR = gadelR + aLeft  * p6 * sqrt(p7)

else
aLeft, aRight diskin2 p8, p9
outs aLeft*p4* sqrt(1-p7), aRight*p4* sqrt(p7)
garbL = garvbL + p5 * aLeft * sqrt(1-p7)
garvbR = garvbR + p5 * aRight * sqrt(p7)
gadelL = gadelL + aLeft  * p6 * sqrt(1-p7)
gadelR = gadelR + aRight  * p6 * sqrt(p7)
endif

endin

</CsInstruments>
```

Differently from the instruments, global effects are all built to run forever (`p3 = -1`) and use channels to manipulate their parameters, instead of using pfields.

Example of a reverb global effect

```
<CsInstruments>

;Reverb

garvbL, garvbR init 0

gkfbrev init 0.4
gkcfrev init 15000
```

```
gkvolrev init 1

gkfbrev chnexport "fbrev", 1, 2, 0.4, 0, 0.99
gkcfrev chnexport "cfrev", 1, 2, 15000, 0, 20000
gkvolrev chnexport "volrev", 1, 2, 1, 0, 1

instr 550 ; ReverbSC

aoutL, aoutR reverbsc garvbL, garvbR, gkfbrev, gkcfrev
outs aoutL * gkvolrev ,  aoutR * gkvolrev
clear garvbL, garvbR

endin

</CsInstruments>
```

3 Haskell Library

3.1 Data Structures

The ensemble of instruments contained in `kairos.csd` gets triggered with instructions coming from the Kairos Haskell library. This is the part of the software responsible for scheduling score events, sending them to Csound at the appropriate time and changing parameters of global effects.

All of the instruments and effects are represented in Haskell using the `Instr` data type, which not only has information about the instrument number and pfields of an instrument, but also about its status (active or inactive), the current note to be played and the patterns of parameters for every pfield.

All of the instruments are collected in a data structure called `Orchestra` that holds them and associates every instrument with a string that identifies it's name.

The two preceding instruments represented in Haskell with their Orchestra

```
sampler :: String -> IO Instr
sampler path = do
  pfields <- newTVarIO $ M.fromList [(3,Pd 1),(4,Pd 1)
                                    ,(5,Pd 0),(6, Pd 0)
                                    ,(7,Pd 0.5),(8,Ps path)
                                    ,(9,Pd 1)] -- p8 : Sample path, p9 : pitch
  emptyPat <- newTVarIO M.empty
  return $ I { insN   = 1
             , pf     = pfields
             , toPlay = Nothing
             , status = Stopped
             , timeF = ""
```

```
              , pats = emptyPat
              }

reverb :: IO Instr
reverb = do
  pfields <- newTVarIO $ M.fromList [(3,Pd (-1))]
  emptyPat <- newTVarIO M.empty
  return $ I { insN   = 550
             , pf     = pfields
             , toPlay = Nothing
             , status = Stopped
             , timeF = ""
             , pats = emptyPat
             }

defaultOrc :: IO Orchestra
defaultOrc = do
  k <- sampler "/KairosSamples/kicks/Kick909.wav"
  rev  <- reverb
  orc  <- atomically $ newTVar $ M.fromList [("K909",k) ,("rev",rev)]
  return $ orc
```

The `Orchestra` is held in a container named `Performance` that holds the Orchestra and the informations about tempo, time signatures and rhythmic patterns readily available to be used by the instruments.

The library is designed to be used within the `GHCi` [7] environment and can be loaded and started running `:script BootKairos.hs` from within GHCi, launching it from the folder containing the script. This script also sets up a number of convenient functions that help compose and modify patterns of instructions easily.

3.2 Operational principles

To use the library, first start the Csound server running the file `kairos.csd` and then launch an instance of GHCi and run the script `BootKairos.hs`. This script will load the necessary modules of the library, run all of the necessary setup steps to start a new `Performance` and also load many functions designed to reduce the amount of typing necessary to perform and simplify the interaction with the Csound orchestra.

To play an instance of an instrument a rhythmic pattern must be assigned to it and then start the play loop.

Playing a four on the floor pattern with the kick instrument from before

```
Kairos>  cPat "fourFloor" "K909" >> p "K909"
```

Patterns of values can then be assigned to the exposed synthesis parameters of the instrument. Every one of this pattern of values gets assigned an update function that determine in which way the value for that parameter will be picked for the next score event. The function can be picking a value from the list or modify the list itself.

Changing the panning and volume parameters

```
Kairos>  vol "K909" [Pd 1, Pd 0.8, Pd 0] randomize
Kairos>  pan "K909" [Pd 0, Pd 1] nextVal
```

An alternative option is the params function, that allows to declare and assign multiple parameters of pfields at the same time.

An example of the params function

```
Kairos> params "K909" [(keep, vol, [Pd 1]),(randomize, pan, toPfD [0, 1])]
```

4 Future Directions

The author uses the library in performances of live dance music and in a experimental electronic band context in a setup with Eurorack modular synthesizers.

One of the features that will soon be introduced is the ability to have a shared clock between multiple instances of Kairos to allow for ensemble performances.

On the musical side, the current focus of the research is on how to more effectively generate and manipulate streams of pfields in interesting ways. The author is now focusing on fractal-based models, Markov chains and autonomous agents.

References

1. Kairos Github repository, `https://github.com/Leofltt/Kairos`
2. Csound UDP Server, `https://csound.com/docs/manual/udpserver.html`
3. SuperCollider `https://supercollider.github.io/`
4. McLean, A. and Wiggins, G.: Tidal - Pattern Language for the Live Coding of Music. In: Proceedings of the 7th Sound and Music Computing conference (2010)
5. Bell, R.: An Interface for Realtime Music Using Interpreted Haskell (2011)
6. Sonic Pi `https://sonic-pi.net/`
7. GHC/GHCi HaskellWiki, `https://wiki.haskell.org/GHC/GHCi`