The Arpeggigon: A Functional Reactive Musical Automaton

Demo, FARM 2017, 9 Sept., Oxford

Henrik Nilsson

Joint work with Guerric Chupin and Jin Zhan

Functional Programming Laboratory, School of Computer Science

University of Nottingham, UK

The Arpeggigon

- Software realisation of the reacTogon:

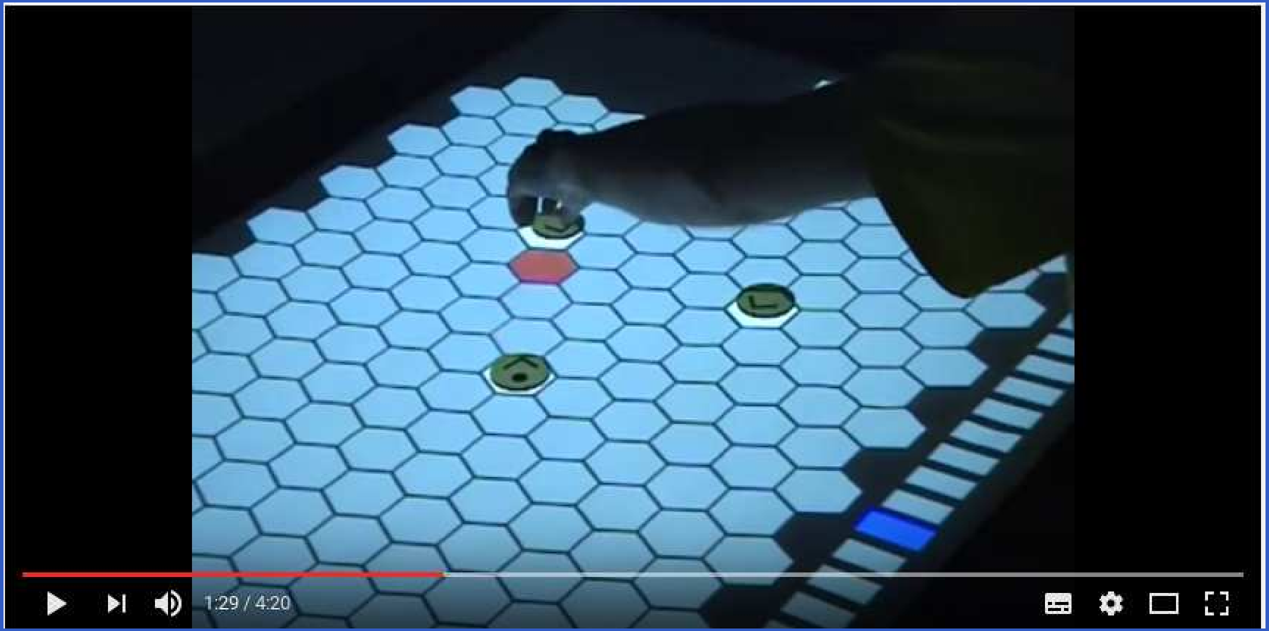
The Arpeggigon

- Software realisation of the reacTogon:

- Interactive cellular automaton:
  - Configuration
  - Performance parameters

The Arpeggigon

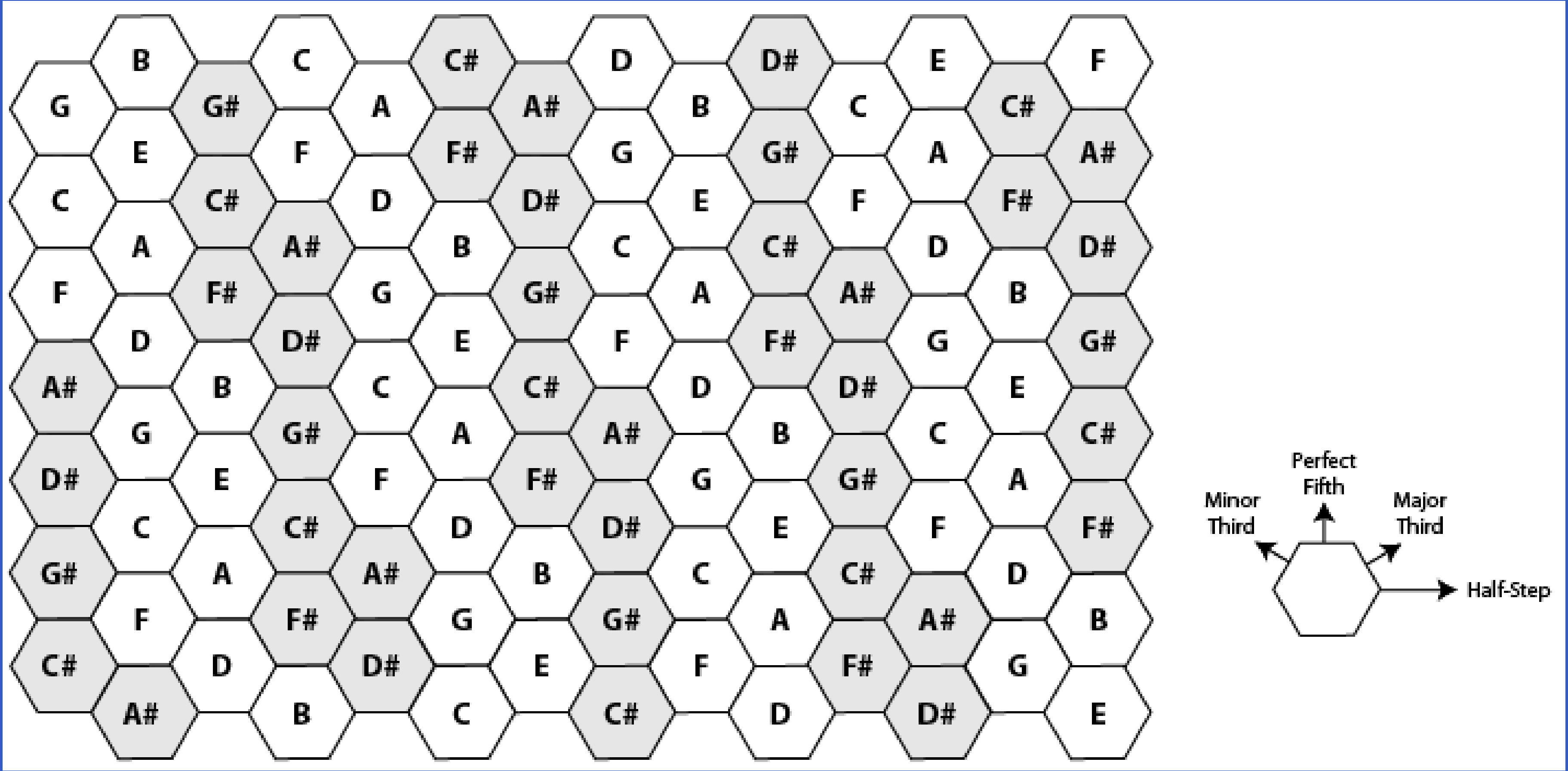
- Software realisation of the reacTogon:



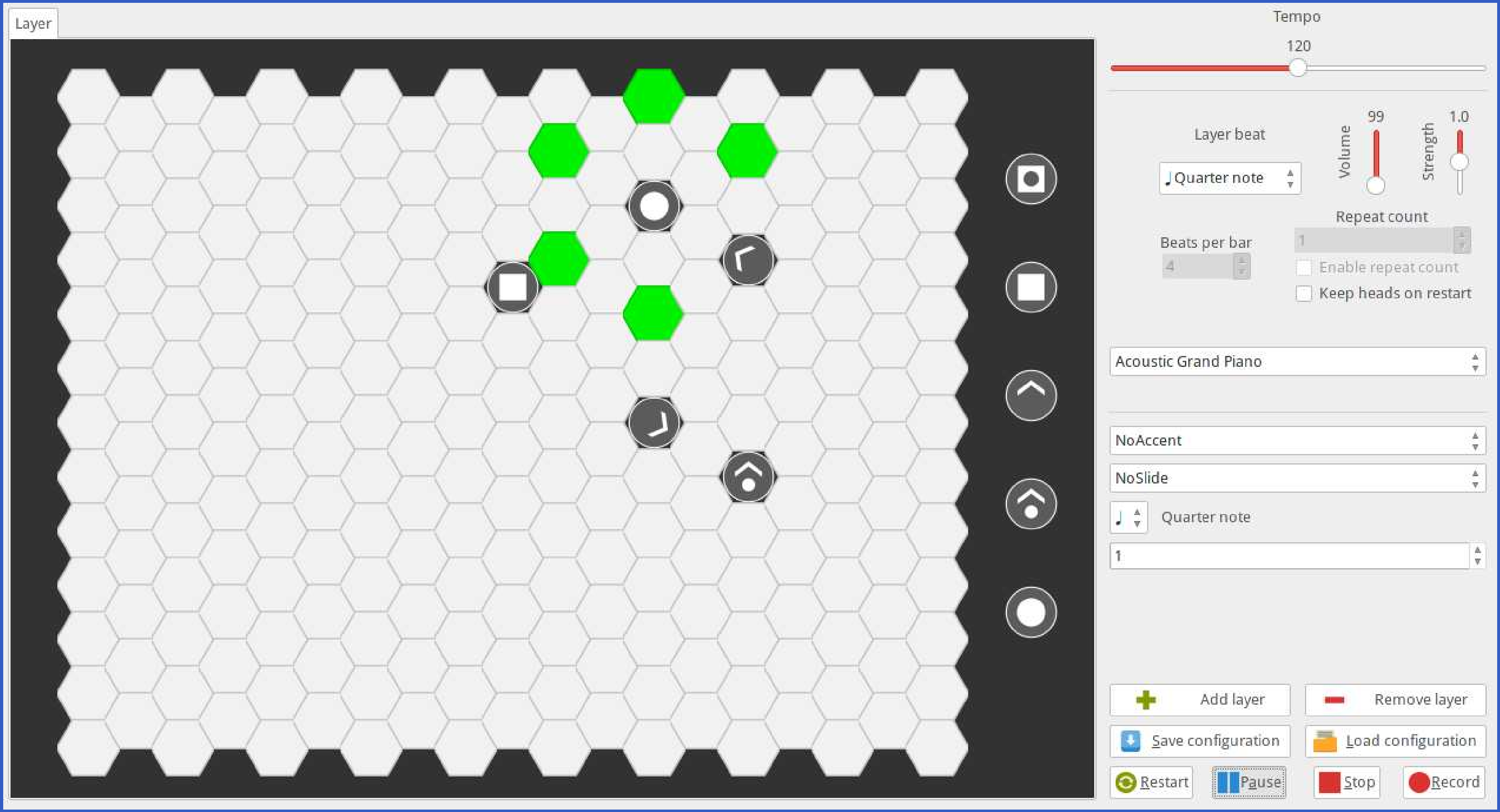
- Interactive cellular automaton:
  - Configuration
  - Performance parameters

Before you get too excited: ***Work in progress!***

The Harmonic Table



Running a Sample Configuration



Tempo: 120

Layer beat: Quarter note

Volume: 99

Strength: 1.0

Repeat count: 1

Beats per bar: 4

Enable repeat count

Keep heads on restart

Acoustic Grand Piano

NoAccent

NoSlide

Quarter note

1

Motivation

Exploring FRP and RVR as an (essentially) declarative way for developing full-fledged musical applications:

- FRP aligns with declarative and temporal (discrete and continuous) nature of music
- RVR allows declarative-style interfacing with external components

Motivation

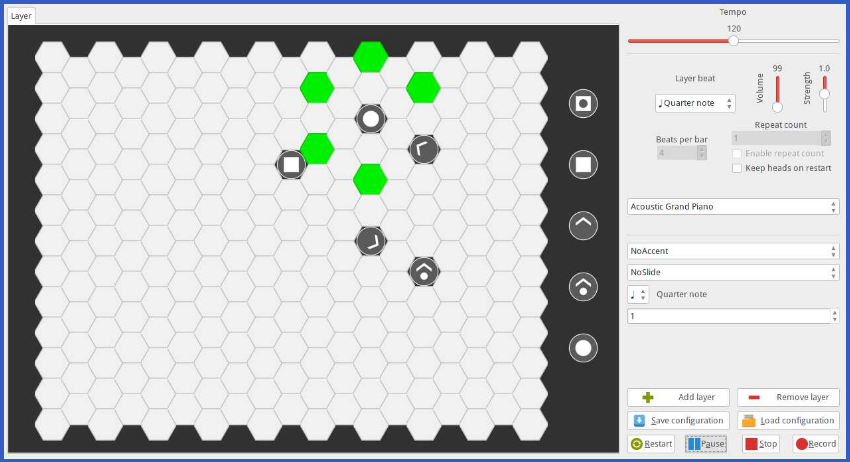
Exploring FRP and RVR as an (essentially) declarative way for developing full-fledged musical applications:

- FRP aligns with declarative and temporal (discrete and continuous) nature of music
- RVR allows declarative-style interfacing with external components

Rest of talk:

- Demonstration
- Implementation Highlights

Aspects of the Arpeggigon (1)



Tempo: 120

Layer beat: Quarter note

Volume: 99

Strength: 1.0

Repeat count: 1

Beats per bar: 4

Enable repeat count

Keep heads on restart

Acoustic Grand Piano

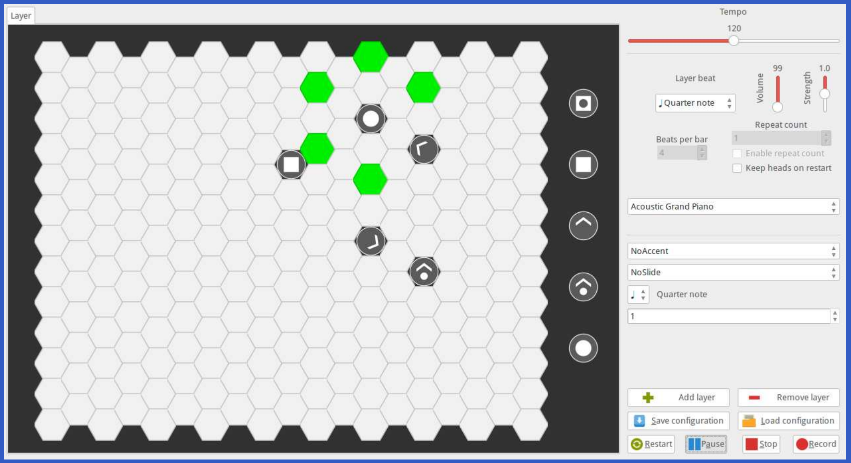
NoAccent

NoSlide

Quarter note

1

Aspects of the Arpeggigon (1)


Tempo 120

Layer beat: Quarter note

Volume 99

Strength 1.0

Beats per bar 4

Repeat count 1

Enable repeat count

Keep heads on restart

Acoustic Grand Piano

NoAccent

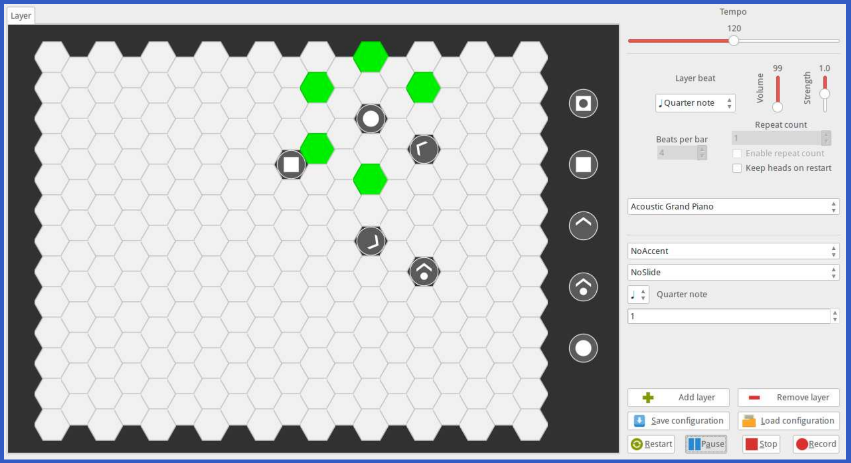
NoSlide

Quarter note

1

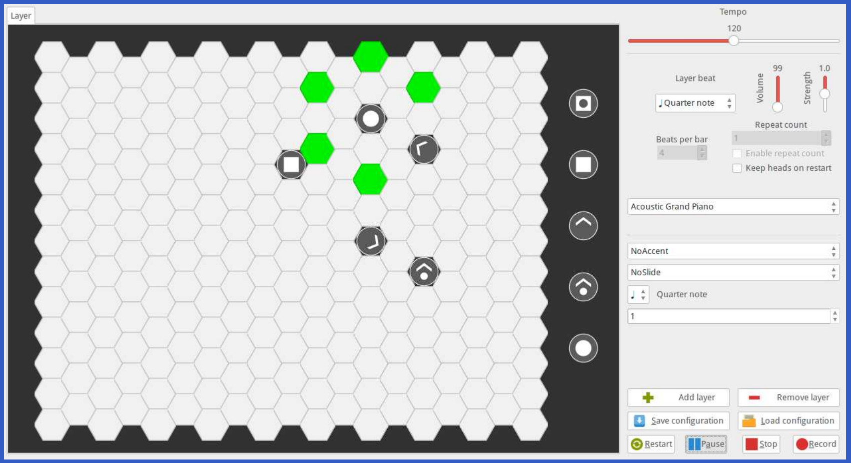
- ***Interactive***

Aspects of the Arpeggigon (1)



- ***Interactive***
- Layers can be added/removed: ***dynamic structure***

Aspects of the Arpeggigon (1)


Layer

Tempo 120

Layer beat: Quarter note

Volume 99

Strength 1.0

Beats per bar 4

Repeat count 1

Enable repeat count

Keep heads on restart

Acoustic Grand Piano

NoAccent

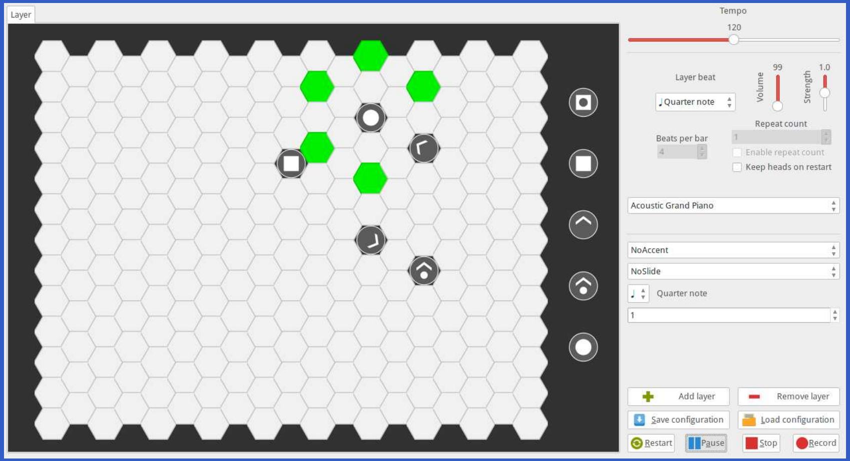
NoSlide

Quarter note

1

- ***Interactive***
- Layers can be added/removed: ***dynamic structure***
- Notes generated at ***discrete*** points in time

Aspects of the Arpeggigon (1)



Tempo: 120

Layer beat: Quarter note

Volume: 99

Strength: 1.0

Repeat count: 1

Beats per bar: 4

Enable repeat count

Keep heads on restart

Acoustic Grand Piano

NoAccent

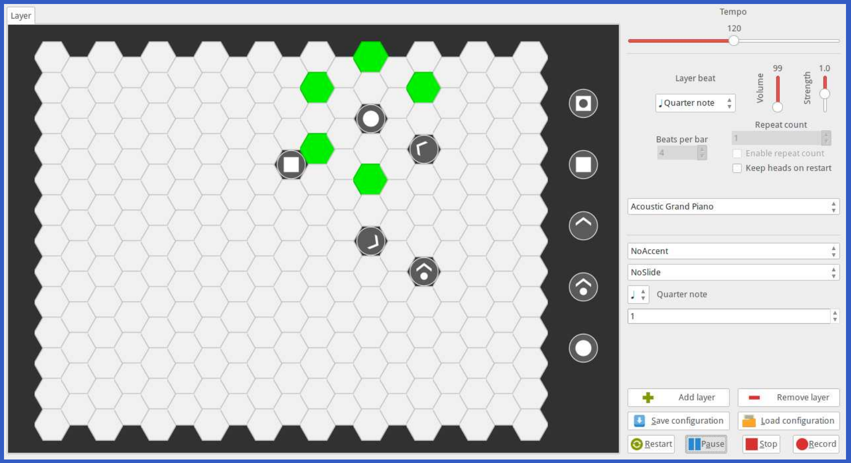
NoSlide

Quarter note

1

- *Interactive*
- Layers can be added/removed: *dynamic structure*
- Notes generated at *discrete* points in time
- Notes played *slightly shorter* than nominal length

Aspects of the Arpeggigon (1)



Tempo: 120
Layer beat: Quarter note
Volume: 99
Strength: 1.0
Beats per bar: 4
Repeat count: 1
Enable repeat count
Keep heads on restart
Acoustic Grand Piano
NoAccent
NoSlide
Quarter note
1

- ***Interactive***
- Layers can be added/removed: ***dynamic structure***
- Notes generated at ***discrete*** points in time
- Notes played ***slightly shorter*** than nominal length
- Configuration and performance parameters can be changed at ***any*** time

Aspects of the Arpeggigon (2)

Potential further enhancements, e.g.:

- Swing: alternately lengthening and shortening pulse divisions
- Staccato and legato playing
- Sliding notes
- Automated, smooth, performance parameter changes

Yampa

Something like ***Yampa*** a good fit:

Yampa

Something like ***Yampa*** a good fit:

- FRP implementation embedded in Haskell

Yampa

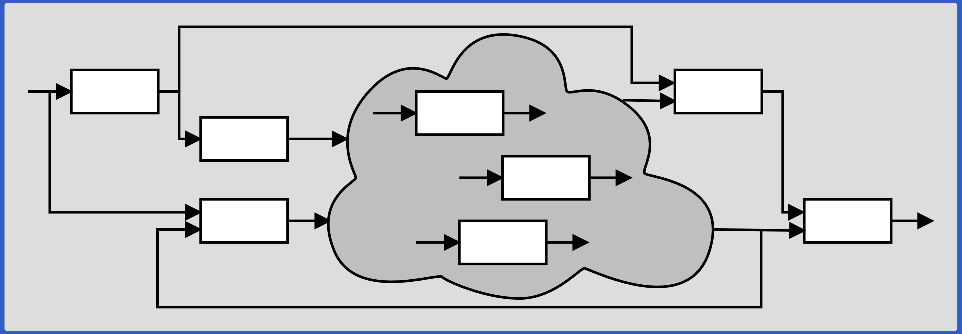
Something like ***Yampa*** a good fit:

- FRP implementation embedded in Haskell
- Supports:
  - ***Signal Functions***: pure functions on signals
  - Structural change through ***Switching***
  - ***Hybrid*** (continuous and discrete) time.

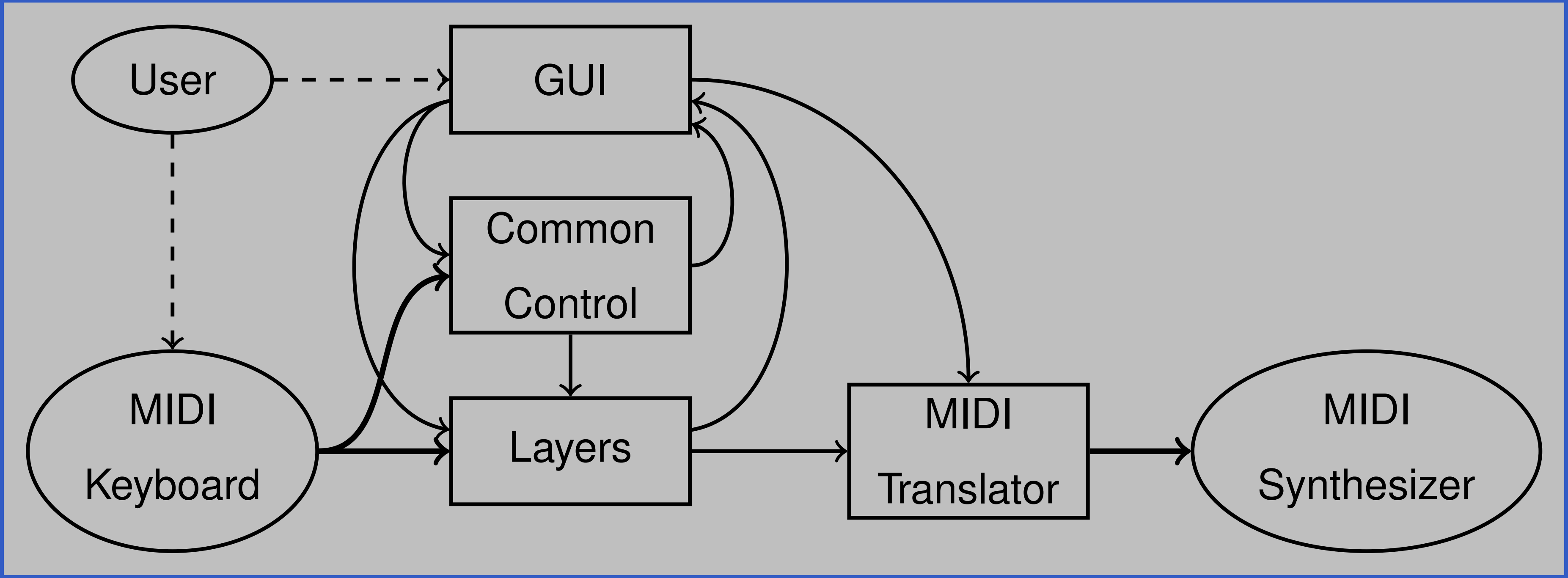
Yampa

Something like ***Yampa*** a good fit:

- FRP implementation embedded in Haskell
- Supports:
  - ***Signal Functions***: pure functions on signals
  - Structural change through ***Switching***
  - ***Hybrid*** (continuous and discrete) time.
- Programming model:



Arpeggigon Architecture



Cellular Automaton

State transition function for the cellular automaton:

$$\begin{aligned} \mathit{advanceHeads} :: \mathit{Board} \rightarrow \mathit{BeatNo} \rightarrow \mathit{RelPitch} \rightarrow \mathit{Strength} \\ \rightarrow [\mathit{PlayHead}] \rightarrow ([\mathit{PlayHead}], [\mathit{Note}]) \end{aligned}$$

Lifted into a signal function primarily using $\mathit{accumBy}$:

$$\mathit{accumBy} :: (b \rightarrow a \rightarrow b) \rightarrow b \rightarrow \mathit{SF}\ (\mathit{Event}\ a)\ (\mathit{Event}\ b)$$

$$\begin{aligned} \mathit{automaton} :: [\mathit{PlayHead}] \\ \rightarrow \mathit{SF}\ (\mathit{Board}, \mathit{DynamicLayerCtrl}, \mathit{Event}\ \mathit{BeatNo}) \\ (\mathit{Event}\ [\mathit{Note}], [\mathit{PlayHead}]) \end{aligned}$$

Automated Smooth Tempo Change

Smooth transition between two preset tempos:

```
smoothTempo :: Tempo → SF (Bool, Tempo, Tempo, Rate) Temp
smoothTempo tpo0 = proc (sel1, tpo1, tpo2, rate) → do
  rec
    let desTpo = if sel1 then tpo1 else tpo2
        diff   = desTpo − curTpo
        rate'  = if      diff > 0.1  then rate
                 else if diff < −0.1 then − rate
                 else                     0
    curTpo ← arr (+tpo0) ⋘ integral ≺ rate'
  returnA ≺ curTpo
```

Reactive Values and Relations (1)

- The Arpeggigon interacts with the outside world using two imperative toolkits:
  - GUI: GTK+
  - MIDI I/O: Jack

Reactive Values and Relations (1)

- The Arpeggigon interacts with the outside world using two imperative toolkits:
  - GUI: GTK+
  - MIDI I/O: Jack
- Very imperative APIs: Hard or impossible to provide FRP wrappers.

Reactive Values and Relations (1)

- The Arpeggigon interacts with the outside world using two imperative toolkits:
  - GUI: GTK+
  - MIDI I/O: Jack
- Very imperative APIs: Hard or impossible to provide FRP wrappers.
- Instead, we use ***Reactive Values and Relations*** (RVR) to wrap the FRP core in a "shell" that acts as a bridge between the outside world and the pure FRP core.

Reactive Values and Relations (2)

- A Reactive Value (RV) is a typed mutable value with access rights and subscribable change notification.

Reactive Values and Relations (2)

- A Reactive Value (RV) is a typed mutable value with access rights and subscribable change notification.
- RVs provide a uniform interface to GUI widgets, files, network devices, . . .

  For example, the text field of a text input widget becomes an RV.

Reactive Values and Relations (2)

- A Reactive Value (RV) is a typed mutable value with access rights and subscribable change notification.
- RVs provide a uniform interface to GUI widgets, files, network devices, . . .

  For example, the text field of a text input widget becomes an RV.
- Reactive Relations (RR) allow RVs to automatically be kept in synch by specifying the relations that should hold between them.

System Tempo Slider

```
globalSettings :: IO (VBox, ReactiveFieldReadWrite IO Int)
globalSettings = do
  globalSettingsBox ← vBoxNew False 10
  tempoAdj          ← adjustmentNew 120 40 200 1 1 1
  tempoLabel        ← labelNew (Just "Tempo")
  boxPackStart globalSettingsBox tempoLabel PackNatural 0
  tempoScale        ← hScaleNew tempoAdj
  boxPackStart globalSettingsBox tempoScale PackNatural 0
  scaleSetDigits tempoScale 0
  let tempoRV =
      bijection (floor, fromIntegral)
      `liftRW` scaleValueReactive tempoScale
  return (globalSettingsBox, tempoRV)
```

Summary

- Yampa (FRP) good fit for writing interactive musical applications in a declarative way.
- Reactive Values and Relations proved very helpful for bridging the gap between the outside world and the FRP core in a fairly declarative way.
- Performance in terms of overall execution time and space perfectly fine; timing must be improved.
- Musical?

Code: `https://gitlab.com/chupin/arpeggigon`