Modality
modal control in SuperCollider

Jeff Carey, Alberto de Campo,

Hannes Hölzl, Robert van Heumen,

Marije Baalman, Till Bovermann, Miguel Negrão

What is Modality?

- a loose collaboration between a couple of SuperCollider developers and users
- goal is to create a toolkit to make it easy to hook up controllers to sound in SuperCollider
- do complicated mapping
- change mappings on the fly, while playing the instrument (modal control)

What is Modality?

- a loose collaboration between a couple of SuperCollider developers and users
- goal is to create a toolkit to make it easy to hook up controllers to sound in SuperCollider
- do complicated mapping
- change mappings on the fly, while playing the instrument (modal control)

First meeting between Jeff Carey, Bjørnar Habbestad, Alberto de Campo, Wouter Snoei and Marije Baalman in October 2010 in Bergen (BEK).

Second meeting here at STEIM with Jeff Carey, Alberto de Campo, Marije Baalman, Till Bovermann, Miguel Negrão, Hannes Hölzl and Robert van Heumen.

Residency

- Presentation and discussion at SC-user meeting
- first three days presentations of each others work
- and a lot of discussion
- and brainstorming
- Then coding, coding, coding
- and brainstorming
- and playing a concert on Monday (DNK)
- some more coding and discussion
- and creating some instruments for tonight.

Concept

- Support different devices that work with different protocols (MIDI, HID, OSC, Serial, etc)
- Provide a common interface to use these devices
- Provide a system to process the data from these devices
- Make real and virtual interfaces interchangeable (GUI for device, processed data for device, etc)

Realisation

- Support different devices that work with different protocols (MIDIMKtl, HIDMKtl)
- Provide a common interface to use these devices (MKtl)
- Provide a system to process the data from these devices (Dispatch)
- Templates for various devices
- Templates for various common ways of processing
- Same interface for MKtl and Dispatch

What took you so long?

- Different protocols have different ways of transporting the data
- Different operating systems (well, Linux and OSX) provide data in different ways
- Semantics of different controllers

A general solution is not so trivial.

Device description index

```
IdentityDictionary[
\nanoKONTROL ->
(osx: ( device: "nanoKONTROL"),
linux: ( device: "nanoKONTROL-nanoKONTROL MIDI 1"),
protocol: \midi, file: "nanoKONTROL.desc.scd" ),
\GamePad -> ( type: \template, protocol: \hid,
file: "GamePad.desc.scd" ), // gamepad template
\Run_N_Drive ->
(osx: ( device: "Run'N' Drive"),
linux: (device: "Thrustmaster Run\'N\' Drive"),
protocol: \hid, file: "Run_N_Drive.desc.scd" ),
\manta ->
(inport: 1234, outport: 5678, protocol: \osc,
file: "Manta.desc.scd" )
]
```

Device description

```
// right hand side four labeled buttons
\bt1r, (type: \button, osx: (cookie: 2), linux: (slot: [1,304]),
  spec: \hidBut, mode: \push),
\bt2r, (type: \button, osx: (cookie: 3), linux: (slot: [1,305]),
  spec: \hidBut, mode: \push),

// joystick axes switches
\joyLX, (type: \joyAxis, osx: (cookie: 15, spec: \cent255inv),
    linux: (slot: [3,0], spec: \cent1 ),  mode: \center),
\joyLY, (type: \joyAxis, osx: (cookie: 16, spec: \cent255 ),
    linux: (slot: [3,1], spec: \cent1 ),  mode: \center),
```

Dispatching

- Each control can cause a chain of actions
- A unit of actions that provide the calculations or logic to create new outputs is called a **Dispatch**
- They can be handwritten
- Or created from a template

Dispatching

- Each control can cause a chain of actions
- A unit of actions that provide the calculations or logic to create new outputs is called a **Dispatch**
- They can be handwritten
- Or created from a template

    - trigger
    - paged
    - threshold
    - thresholdUp, thresholdDown
    - up, down
    - thresholdZone
    - merge
    - multiclick

Dispatch template example

```
(
func:{ |disp, source,elemKeys, sourceKey|
disp.map(source, elemKeys);
disp.createOutputsFromInputs;

disp.addToProc( \trigger, { |dis,e|
var in = dis.changedIn;
if( in[\val] == 1 ) {
dis.setOutput(in[\key], 1)
}
});
   disp
},
desc: "trigger on a value of 1",
name: "trigger",
type: "creator"
)
```

A simple instrument

```
MKtl.find;
a = MKtl.new( ’ngms0’ );

Ndef( \sine, { Pan2.ar( Mix.new( SinOsc.ar(
  [\freq1.kr(400).lag(0.3,0.5), \freq2.kr(400).lag(0.3,0.5),
  \freq3.kr(400).lag(0.3,0.5)]  * [3/4,1,4/3] ) ) / 10,
  \pos.kr(0) ) } );
Ndef( \sine ).fadeTime = 0.1;

d = Dispatch.new( \trigger, a,
      a.elements.select{ |it| it.type == \button }.collect( _.name ) );

d.addToOutput( \btgreen, \playSynth, { Ndef(\sine).play( fadeTime: 1 ); } );
d.addToOutput( \btred, \stopSynth, { Ndef(\sine).stop(1); } );

a.addToOutput( \accX, \changeFreq, { |c| Ndef( \sine ).set( \freq1,
      [400,2000,\exponential].asSpec.map( c.value ) ) } );
a.addToOutput( \accY, \changeFreq, { |c| Ndef( \sine ).set( \freq2,
      [400,2000,\exponential].asSpec.map( c.value ) ) } );
a.addToOutput( \accZ, \changeFreq, { |c| Ndef( \sine ).set( \freq3,
      [400,2000,\exponential].asSpec.map( c.value ) ) } );
```