IPyCLAM
Enpowering CLAM with Python

David García Garzón
(UPF, CLAM Project)
Xavier Serra Román
(Dolby, CLAM Project)

http://clam-project.org

LAC2013 Graz
Outline

- Introduction to CLAM
- API
- Engines (CLAM, JACK...)
- Prototyping
- Conclusions
The CLAM project

- Born at Universitat Pompeu Fabra, 2001
- Adopted by Barcelona Media Foundation, 2007
- Startups, acquisitions by big corporations...
  - Team members busy
  - Potential contributions won't be released
- Nowadays, it has no support from any parent institution like it had in the past.
Buried? Not entirely!!

- A bunch of developers still push in their spare time.
- Big project, few hands...
- Wanna join?
CLAM
CLAM: building blocks
CLAM: visual prototyping
CLAM: visual prototyping
Why Python?

- Fast development
- Interactive
- But... wasn't Python unsafe for real-time?
  - Nevermind, RT code is isolated inside modules
  - Let Python play the glue role
How does IPyClam empower CLAM?

- Powerful prototyping language
  - PySide/PyQt4
- Interactive manipulation of networks
- Serialization format
- Parametric networks
A not so complex network
API design goals

- Do not mimic C++ API
- Python expressiveness
  - Slices, dynamic attributes, iterators...
- Redundant API:
  - Offer the convenient API but also the API that being less convenient cover all cases.
- Interactive use:
  - Object discovery by tab completion
Convenience vs. versatility

- **Convenient way**
  net.processing1.port1
  - Short and enables tab completion discovery

- **Most versatile way**
  net["processing1"].inports["port1"]
  - Invalid Python identifiers
  - Collisions with existing methods/attributes
  - Collisions with outports/controls/configs
An example: JACK stereo wire

```python
from ipyclam import Network
n = Network()
n.source = "AudioSource"
n.sink = n.types.AudioSink
n.source.NSources = 2
n.sink.NSinks = 2
n.source > n.sink
n.backend = "JACK"
n.play()
```
Module creation

• Assign a new attribute or item
  
  n.newproc = ...
  n["newproc"] = ...

• To a string
  
  n.newproc = "AudioSource"

• Or to a member of n.types.
  
  n.newproc = n.types.AudioSources

• Provides available types by tab completion
Module configuration

- Attribute or item assignment
  
  net.myprocessing.parameter = “value”
  net.myprocessing['parameter'] = “value”
  net.myprocessing.config.config.parameter = “value”

- Holding reconfiguration
  
  with net.myprocessing.config as c :
      c.parameter1 = 1000
      c.parameter2 = 2000
Connections: Broadcasting

- One to one
  net.source.outport1 > net.sink.inport1
- One to many
  net.source.outport1 > net.sink
- Many to many
  net.source > net.sink
Connections: Slices

- Connecting intervals
  \[ \text{net.source}[2:7] > \text{net.sink} \]
- Connecting just even ports
  \[ \text{net.source}[:2] > \text{net.sink} \]
- Inverting channel order
  \[ \text{net.source}[::-1] > \text{net.sink} \]
Iterables

- Iterable objects:
  
  ```python
  porttypes = {
      port.name: port.type
      for port in net.myproc.outports
  }
  ```

- `net.proc.outports`
- `net.proc.inports`
- `net.proc.outcontrols`
- `net.proc.incontrols`

- `net.processings`
- `net.types`
- `net.proc.port.peers`
- `net.proc.config`
Audio backends and transport

- Setting the backend property
  ```javascript
  net.backend = "PortAudio"
  ```
- Controlling the playback
  ```javascript
  net.play(), net.stop(), net.pause()
  net.isPlaying(), net.isStopped(), net.isPaused()
  ```
Self replicable

- net.code() generates the code needed to regenerate itself.
- Alternative to current XML serialization
- More readable
- Not safe if using the Python interpreter!!
- Fast display: If you just type 'net' prints the code.
Integrated console

In [19]: net.sink="AudioSink"

In [20]: net.source="AudioSource"

In [21]: net.sink.NSinks = 4

In [22]: net.source.NSources = 4

In [23]: net.source._outports[:] > net.sink._imports[:]-1

In [24]:
JACK engine, ¿IPyJack?

```python
In [1]: from ipyclam.Jack_Engine import Jack_Engine
In [2]: from ipyclam.Network import Network
In [3]: j=Network(Jack_Engine())

In [4]: j
Out[4]:
network.system = 'JackClient'
network.FilePlayer = 'JackClient'
network.Hydrogen = 'JackClient'

network.Hydrogen.out_L > network.system.playback_1
network.Hydrogen.out_R > network.system.playback_2

In [5]: j.FilePlayer > j.system
Out[5]: 1

In [6]: j.FilePlayer > j.system._imports[1]
Out[6]: 1

In [7]: j
Out[7]:
network.system = 'JackClient'
network.FilePlayer = 'JackClient'
network.Hydrogen = 'JackClient'

network.FilePlayer.AudioSink_0 > network.system.playback_2
network.FilePlayer.AudioSink_0 > network.system.playback_1
network.Hydrogen.out_L > network.system.playback_1
network.Hydrogen.out_R > network.system.playback_2
```
How?

- Original intent: decouple syntactic sugar from the code that does stuff. Mock-ups.
- Side effect: Reimplementing the engine API for a different system, like JACK is fast!
PySide/PyQt4 integration
import QtGui from PySide
import ipyclam.ui.PySide as ui

app = QtGui.QApplication(sys.argv)
network = ipyclam.Network()
network.load("sms.clamnetwork")
window = ui.loadUi("dialog.ui")
network.bindUi(window)
window.show()
network.play()
app.exec_()
A simple oscilloscope

- Creating widgets with Qt factories
- Assigning binding properties:
  
  ```python
  net.source = "AudioSource"
  w = ui.createWidget("Oscilloscope")
  w.setProperty("clamOutport", "source.1")
  net.bindUi(w)
  w.show()
  ...
  ```
Conclusions

- Nice API!
- Reusable for other systems like JACK
- Prototyping: Qt + Python + CLAM
- Integrated console for interactive manipulation and exploration of networks.
Future work

- Fixing NetworkEditor interaction:
  - Canvas update.
  - Processing placement
- Examples, examples, examples.
- Numpy based audio backend
- Modules in Python for offline processing
- Other engines: gAlan, Patchage...
Questions?
Thanks