COSC 4P98 Lecture notes: **Music Programming Languages**
Feb 24, 2014
B. Ross

- Many specialized programming environments and languages for music composition and audio processing.
- Wikipedia pages:
    1. http://en.wikipedia.org/wiki/Comparison_of_audio_synthesis_environments
    2. http://en.wikipedia.org/wiki/Audio_programming_language
- Some critieria for evaluating them include:
    1. Usability: technical skill needed to program them
    2. Learnability: learning curve
    3. Sound quality (subjective?)
    4. Creative workflow: does it promote a creative process?
    5. Performance: CPU usage, bandwidth, latency, concurrency
    6. Stability
    7. Support: user community, documentation, tutorials
    8. Capabilities: what it can do (realtime?)
    9. Integration with other systems
- Some languages are library extensions to conventional languages (C++, Python, Java)
- Others are standalone systems, developed with music processing in mind.

- Some examples (of many!):

1. **CSound**

    - first released in 1986
    - Large user community.
    - Possibly the most powerful audio processing environment around.
    - Many extensions: real-time processing, hooks to other languages
    - Recently a VST version was released.
    - very stable.
    - Learning curve is not too bad, since it is a simple data-flow language.
    - BUT... not as algorithmic as newer systems, and therefore not very suitable for algorithmic composition. (But there are now extensions... see 2 below).
    - We will look at it in detail (see CSound notes and lectures).

2. **CSoundAC**

- Python extension included in latest CSound distribution
- Permits Python statements to be used for generating CSound files.
- Opens up CSound to algorithmic composition.
- Python is stable, easy to learn.
- Example (from CSoundAC tutorial document).

```
r = 3.974
y = 0.5
time_ = 0.0
duration = 0.25
istatements = [ ]

for i in xrange(1000):
        y = r * y * (1.0 - y)
        time_ = time_ + duration / 2.0
        midikey = int(36.0 + (y * 60.0))
        istatement = "i 1 %f %f %d 80\n" %
        (time_, duration, midikey)
        print istatement,
        istatements.append(istatement)

score = string.join(istatements)
```

- This code implements a “strange attractor”. As j iterates, y either converges, oscillates, or never terminates. The code (in Python) generates lines for a CSound score file.
- See the complete example in the tutorial file.
- If you have an algorithm for generating notes (say), this Python extension lets you implement that algorithm. The computed notes or other parameters are then written to standard CSound score files.
- Without this, one would have to either manually write the score file (unlikely!) or have another program do it, and then translate it into CSound syntax.

References:

- Tutorial by M. Gogins: https://www.dropbox.com/s/0d7rxjy7pqlx5w2/tutorial.zip
  local Brock copy: http://www.cosc.brocku.ca/Offerings/4P98/local/tutorial.zip
- http://www.linuxjournal.com/content/introducing-csoundac-algorithmic-composition-csound-and-python

COSC 4P98 Lecture notes: **Music Programming Languages**
Feb 24, 2014
B. Ross

3. **SuperCollider**

- Introduced in 1996.
- Combines object-orientation, functional programming, C syntax
- very powerful environment.
- Real-time audio synthesis.
- Annual (?) conference on it.
- Multi-channel support. Very easy to convert a mono signal to 8 channels (for example).
- Fairly steep learning curve. You need background in programming (OO, Functional languages).
- Live coding: changing/editing code in real-time, to affect performance.
- EXAMPLE:

```
(play(
        {
                CombN.ar(
                        SinOsc.ar(
                                midicps(
                                        LFNoise1.ar(3, 24,
                                                LFSaw.ar([5, 5.123], 0, 3, 80)
                                        )
                                ),
                                0, 0.4),
                        1, 0.3, 2)
        }
))
```

- CombN: comb delay, no interpolation
- "CombN.ar: process at audio sampling rate
- SinOsc: interpolating sine wave table oscillator
- LFNoise1: ramped noise
- LFSaw: sawtooth oscillator
- midicps: convert midi note # to frequency

Multi-channel:

```
{Blip.ar(25, LFNoise0.kr(5, 12, 14), 0.3)}.play // single channel

{Blip.ar(25, LFNoise0.kr([5, 10], 12, 14), 0.3)}.play // stereo

{Blip.ar(25, LFNoise0.kr([5, 10, 2, 25], 12, 14), 0.3)}.play // quad

{Blip.ar(25, LFNoise0.kr([5, 4, 7, 9, 5, 1, 9, 2], 12, 14), 0.3)}.play
// 8
```

COSC 4P98 Lecture notes: **Music Programming Languages**
Feb 24, 2014
B. Ross

More examples: http://supercollider.sourceforge.net/audiocode-examples/

4. **Processing and Beads**

- Begin in 2001.
- Processing has become the programming language of choice of artists and hobbyists (robots, interactive installations,...).
- Implemented in Java. Compilation will create Java code.
- Straight-forward language with high-level control, object-orientation, data types, libraries. All of Java's abilities.
- Interactivity is supported.
- Interfaces to external hardware, such as Arduino boards.
- Lots of audio libraries available
    - p5_sc: interface with SuperCollider
    - MidiBus: a MIDI library

- Beads: realtime audio library
- Library to extend Java to perform audio processing.
- Defines UGens (unit generators), a concept from SuperCollider.
- Beads is a basic library. Not as comprehensive as CSound or SuperCollider.
- Some of the classes:
    - Synth: generate sounds
        - WavePlayer: plays wave data in a buffer
        - Noise: generates white noise
    - Filter:
        - OnePoleFilter: with cutoff freq
        - LPRezFilter: Low-pass filter with resonance
    - Effect:
        - Reverb
    - Sample playback
        - GranularSamplePlayback: granular playback engine

- EXAMPLE

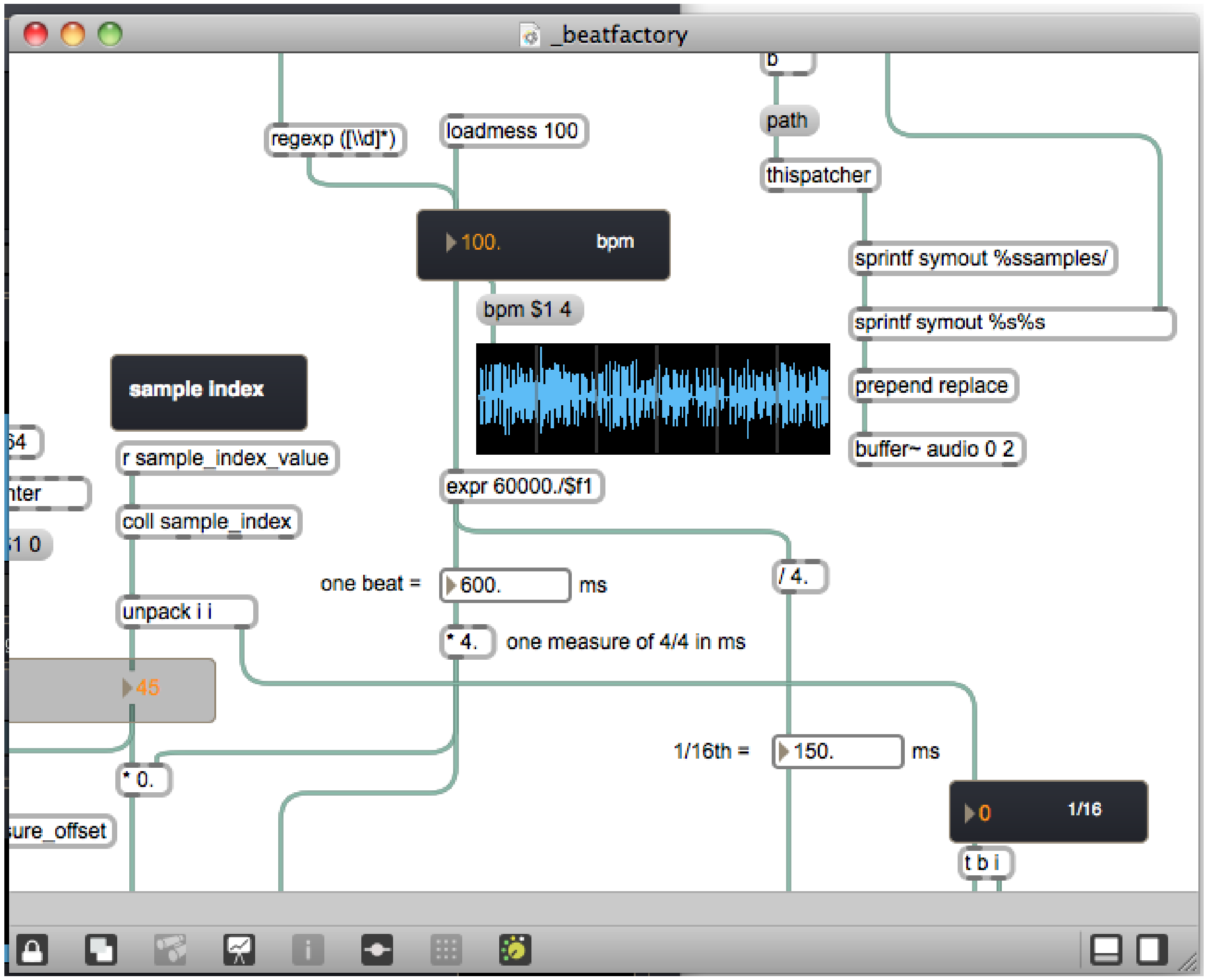
```
  new Bead() {
    //this is the method that we override to make the Bead do something
     public void messageReceived(Bead message) {
        Clock c = (Clock)message;
        if(c.isBeat()) {
          WavePlayer wp = new WavePlayer(ac, (float)Math.random() * 3000 + 100,
Buffer.SINE);
          Gain g = new Gain(ac, 1, new Envelope(ac, 0.1));
          ((Envelope)g.getGainEnvelope()).addSegment(0, 1000, new KillTrigger(g));
          g.addInput(wp);
          ac.out.addInput(g);
       }
     }
```

- http://www.beadsproject.net/examples/Lesson7_Music/applet/index.html

5. Max

- Introduced around 1989.
- Commercial product (Cycling ’74)
- Named after Max Matthews (influential computer musician/scientist).
- Free variants: Pure Data, jMax.
- Visual programming language: Programmer inserts graphical modules, and connects them.
- The connections are equivalent to variables or channels (see CSound instrument definitions).
- New modules:
  - MSP (Max Signal Processing): real-time audio processing
  - Jitter: realtime video, 3D graphics
- Max for Live: integrate Max with Ableton Live. Permits high-level instrument and effecti design, composition tools.

COSC 4P98 Lecture notes: **Music Programming Languages**
Feb 24, 2014
B. Ross

- 

- Example videos:
    - o Monolake granular: http://www.youtube.com/watch?v=9pn_b7OUO6I
    - o Plastikman: http://www.youtube.com/watch?v=PV3pfQFtjSg