JythonMusic: An Environment for Developing Interactive Music Systems

Bill Manaris, Pangur Brougham-Cook
Computer Science Dept.
College of Charleston, USA
manarisb@cofc.edu,
broughamcookpj@g.cofc.edu

Dana Hughes
Computer Science Dept.
University of Colorado, Boulder, USA
dana.hughes@colorado.edu

Andrew R. Brown
Queensland Conservatorium
Griffith University, Australia
andrew.r.brown@griffith.edu.au

ABSTRACT

JythonMusic is a software environment for developing interactive musical experiences and systems. It is based on jMusic, a software environment for computer-assisted composition, which was extended within the last decade into a more comprehensive framework providing composers and software developers with libraries for music making, image manipulation, building graphical user interfaces, and interacting with external devices via MIDI and OSC, among others. This environment is free and open source. It is based on Python, therefore it provides more economical syntax relative to Java- and C/C++-like languages. JythonMusic rests on top of Java, so it provides access to the complete Java API and external Java-based libraries as needed. Also, it works seamlessly with other software, such as PureData, Max/MSP, and Processing. The paper provides an overview of important JythonMusic libraries related to constructing interactive musical experiences. It demonstrates their scope and utility by summarizing several projects developed using JythonMusic, including interactive sound art installations, new interfaces for sound manipulation and spatialization, as well as various explorations on mapping among motion, gesture and music.

Author Keywords

Music and interaction, user interfaces, interactive sound art and installations, musical mapping, Python, software libraries

CCS Concepts

• **Applied computing** → **Sound and music computing**; • **Human-centered computing** → **Interaction design**; • **Information systems** → **Open source software**;

1. INTRODUCTION

Interactive computer music systems emerged in the last 40 years with the development of increasingly more powerful and versatile I/O devices and software frameworks for interaction. These advances in technology, as is usually the case through history, have been exploited in music to develop new and innovative computer-based and electroacoustic music systems, compositions, and interactive experiences [1].

Rowe defines *interactive computer music systems* as music systems that respond to human input, which gives them the ability to participate in live performances [2].

Paine [3] extends this to create a more thorough taxonomy of systems for interactive musical performance, or digital musical interfaces (DMI). He takes into account existing musical instrument taxonomies, such as the Hornbostel and Sachs [4] instrument taxonomy, consisting of *aerophones*, *chordophones*, *idiophones*, *membranophones*, and *electrophones*. He then superimposes the dimensions specified by Birnbaum, et al. [5]. Finally, Paine introduces additional dimensions to form a conceptual map of musical interaction, including **gesture type** (e.g., body movement, eye tracking, senstate surface, steering wheel, etc.), **controller type** (e.g., mobile phone, motion sensor, glove, joystick, digital controller, glass sensor, etc.), and **computer software** used. [3].

Several audio libraries and software packages for musical programming have been developed over the last two decades. In addition to manipulating audio, many focus on other characteristics such as live coding, sample-level manipulation, or minimal programming requirements. Well-established packages include ChucK [6], SuperCollider [7], Pure Data [8], Csound [9], Minim [10], and Gibber [11].

Most of the above packages assume programs are generated or executed in a console, and are not focused on simplifying GUI development in addition to generating audio. Instead, GUI development either relies on packages available in the underlying language (e.g., Java Swing), or a separate GUI development environment (e.g., Qt GUI used by SuperCollider, Tcl/Tk used by Pure Data). These may require learning relatively very complex APIs that expose needless low-level components to the musician (e.g., the Java Swing package contains well over 100 classes, including managers, events, listeners, etc.), or possibly entirely new languages. Alternatively, users may utilize existing front ends. For instance, several front ends have been developed for ChucK since its inception, specifically Audicle [12] and miniAudicle [13, 14]. While these provide a more streamlined interface, and useful visualizations, they still rely heavily on an embedded console for real-time coding, and do not provide a robust set of widgets for building GUI-based musical instruments. Third-party GUI libraries are available for some environments (e.g., for Processing, used by Minim), though these have varying levels of complexity and maintenance.

Finally, modern web browsers allow building cross-platform musical interfaces using web technologies (HTML5, JavaScript, CSS, etc.), while leveraging the inherent layout and networking capabilities of the browser. NexusUI provides an API for UI widget and sending OSC messages via UDP, as well as builders for converting a Max/MSP patch and drag-and-drop interfaces [15]. Similarly, a simple browser-based API for creating and storing interfaces for the Gibberish.js audio library is presented in [16].

This paper presents JythonMusic (http://jythonmusic.org), a software framework for developing interactive musical experiences. JythonMusic is an open-source environment with an extensive online API reference. It has been used to develop various interactive sound art and installations, among other projects, which explore various mapping strategies, and explore relationships among motion, gesture and music.[1]

2. JythonMusic

JythonMusic is based on Python. It was originally developed for teaching computer programming in a musical context [17],

[1] This work has been funded in part by NSF DUE-1323605, DUE-1044861, IIS-1049554, IIS-0849499 and IIS-0736480.

NIME'18, June 3-6, 2018, Blacksburg, Virginia, USA.

based on extensive earlier work in algorithmic music composition [18]. Within the last decade, it has grown to have extensive capabilities for developing music interfaces and interactive experiences across all dimensions mentioned in the previous section. As a result, it may serve as a complement to existing music and other art-related software, such as PureData, Max/MSP and Processing, among many others.

JythonMusic is built on top of jMusic, an extensive music library for music composition and audio processing in Java [19]. It also incorporates other cross-platform libraries, such as jSyn [20], and makes them available via Python syntax.

As an extension of jMusic, it supports computer-aided composition, generative music, instrument building, interactive performance and music analysis. It provides musical data structures (i.e., Note, Phrase, Part and Score), playback of musical scores in real-time, as well as rendering in MIDI or audio for storage and later processing. It can also read and write MIDI files, audio files and XML files, among others. As an extension of jSyn, it provides a modular sound synthesis API based on unit generators, which can be combined to form complex timbres and software synthesizers.

All this functionality is now available through Python. This design choice is very desirable – Python is a general-purpose programming language designed to be succinct and easy to read. Python programs tend to be about three times as short as equivalent programs in Java, and C/C++ [21]. Accordingly, Python has become the most popular introductory programming language at US universities [22]. It is also used extensively by companies such as Google. Finally, Python includes a large and comprehensive set of libraries for common algorithmic tasks.

JythonMusic uses Jython, the version of Python running on top of the Java Virtual Machine. This gives it great portability, as it runs on all popular computing platforms. Additionally, it provides access to the complete Java API through Python syntax, as well as other external Java libraries, as needed.

Finally, through relatively easy-to-use MIDI and OSC libraries, JythonMusic works seamlessly with other music software, such as Pd, Max/MSP, Ableton Live, as the examples shown below demonstrate.

We believe JythonMusic provides a viable alternative to existing systems (see Section 1.1), as it simplifies development of interactive musical experiences. While perhaps not as specialized as some of the above systems, to paraphrase Alan Kay's maxim, it makes simple things simple, and complex things possible. JythonMusic comes with an editor, called JEM, which encapsulates all available libraries, and provides various useful keyboard shortcuts.

2.1 Music Library

The JythonMusic music library provides functionality to aid in transcription, composition, and performance of musical works. It supports both MIDI and audio material.

The Note class treats musical notes as objects, which require a pitch and duration, with a thorough set of constants (such as C4, for middle C, and QN for quarter note). In addition to standard MIDI pitches, the music library supports microtones, for exploring non-traditional tunings, as well as non-Western scales and ancient modes.

Note objects can be added in series to Phrase objects Phrases can then be assembled inside Part objects, which allow instruments to be set. Part objects can be assembled to create Score objects, which can have a title and a tempo, among others. Each of these objects has various setter / getter functions, which are documented online (see http://jythonmusic.org/music-library).

Figure 1 demonstrates a code sample from a laptop orchestra performance of Terry Riley's *In C* (also see section 3.1). Here a temporal recursion pattern is being used, i.e., the loopMusic() function assembles and plays a phrase once and then schedules itself to repeat.

Additional functionality is provided in the Mod class, which contains a large selection of functions for transforming Phrases, Parts, and Scores. The View class contains functions to visually display music in various formats. The music library also supports reading MIDI files as scores and writing scores to MIDI using the Read and Write classes respectively.

Finally, multiple classes are provided for rendering sound. The Play class plays Notes, Phrases, Parts, and Scores using MIDI. The AudioSample class plays / loops audio files, and pitch shifts them in real-time. The LiveSample class can record sounds from a microphone, play / loop them, and pitch shift them in real-time. The Metronome class synchronizes callback functions, which supports live coding tasks, such as building complex rhythmic patterns.

2.2 GUI Library

The JythonMusic GUI library supports development of computer musical instruments and graphical user interfaces, having an extensive set of graphics objects and widgets, with event-driven programming via callback functions, and various types of keyboard and mouse events.

The main GUI object is a Display. A program's GUI exists inside a Display object (window). Displays contain other GUI components (graphics objects and widgets). For example, this:

```
d = Display("Some Display", 400, 100)
```

creates a 400x100 pixel window. No other code is needed, compared to, say, Java Swing, or Tcl/Tk (among others), which require multiple lines of sometimes cryptic code to set up, pack, and render a simple window.

Similarly, creating displayable objects is as simple as creating an object, and then adding to the display – two lines of code. For most,

```
# TerryRiley.InC.py
#
# Live coding performance of Terry Riley's "In C".
# See http://www.flagmusic.com/content/clips/inc.pdf

from music import *
from timer import *

# redefine these notes at will
pitches   = [E4, F4, E4]
durations = [SN, SN, EN]

# play above pitches and durations in a continuous loop
def loopMusic():

   global pitches, durations

   # create phrase from current pitches and durations
   theme = Phrase()
   theme.addNoteList( pitches, durations )

   # play it
   Play.midi( theme )

   # get duration of phrase in millisecs (assume 60BPM)
   duration = int( theme.getBeatLength() * 1000 )

   # create and start timer to call this function
   # once recursively, after the elapsed duration
   t = Timer( duration, loopMusic, [], False )
   t.start()

# start playing
loopMusic()
```

Figure 1. A code excerpt using temporal recursion to perform Terry Riley's *In C*. The JEM editor provides various shortcuts for re-evaluating different sections of running code.

objects, a single-line abbreviation exists. For example, the following adds a circle with a 10-pixel radius to the above display:

```
# x, y, and radius
c = Circle(200, 50, 10)
d.add(c)
```

The GUI library is implemented on top of Java Swing, so it provides access to all Swing functionality. Natively, it offers the following Graphics objects: Line, Circle, Point, Oval, Rectangle, Polygon, Arc, Icon, and Label. Additional GUI control objects include: Button, Checkbox, Slider, DropDownList, TextField, TextArea, and Menu (top, and pop-up). Other widgets for building musical user interfaces include HFader, VFader, Rotary, Toggle, Push, and XYPad.

Finally, the GUI library supports various keyboard and mouse events (such as key press, mouse click, mouse move, and mouse drag) to associate callback functions to handle these events. This allows for developing elaborate, yet easy to comprehend interactive behaviors. For example, Figure 2 displays the interface for a tunable microtonal instrument, for exploring different tuning systems, based on the ancient Greek tetrachord. For more information on the GUI library, see http://jythonmusic.org/gui-library .

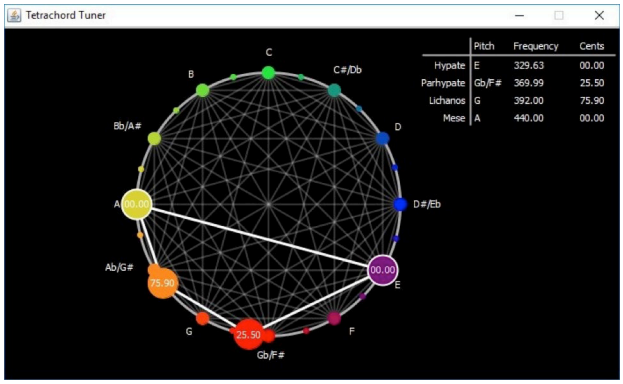

Figure 2. GUI for a microtonal, tetrachord-based tuning system.

3. SOME PROJECTS

This section discusses a few more comprehensive projects developed with JythonMusic, which demonstrate its capabilities and potential for developing interactive musical experiences. They span various categories, including musical interaction, interactive sound and visual art installations, new interfaces for sound manipulation and spatialization, as well as various explorations on mapping among motion, gesture and music.

3.1 A Laptop Orchestra (2010)

In December 2010, we organized a student laptop orchestra performance. Students were introduced to the computer as a musical instrument and as a creative environment to develop fluency with musical practices, including algorithmic composition, developing simple computer instruments, and composing exercises based on models in electroacoustic music and minimalism [18]. Simultaneously, they were introduced to programming in Python. For more information, see http://bit.ly/charlestonLaptopOrchestra .

3.2 Monterey Mirror (2011)

Monterey Mirror [23] is an experiment in interactive music performance. It engages a human performer and a computer (the mirror) in a game of playing, listening, and exchanging musical ideas. The computer player employs an interactive stochastic music generator, which incorporates Markov models, genetic algorithms, and power-law metrics. For more information, see http://bit.ly/montereymirror .

3.3 Time Jitters (2014)

Time Jitters [24] is a four-projector interactive installation (see Figure 3), which was designed by Los Angeles-based visual artist Jody Zellen for the Halsey Institute of Contemporary Art in Charleston, SC, USA.

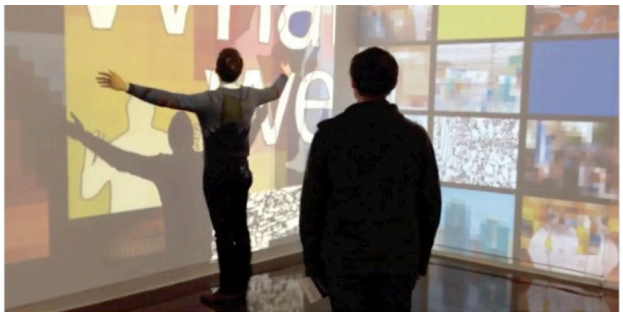

Figure 3. Users interacting with *Time Jitters* at the Halsey Institute of Contemporary Art on opening night.

Time Jitters includes two walls displaying video animation, and two walls with interactive elements. The concept is to create an immersive experience for participants, which confronts them with a bombardment of visual and sound elements. For more information, see http://bit.ly/timeJitters .

3.4 Diving into Infinity (2015)

Diving into Infinity [25] is a Kinect-based system which explores ways to interactively navigate M.C. Escher's works involving infinite regression. It focuses on *Print Gallery*, an intriguing, self-similar work created by M.C. Escher in 1956.

The interaction design allows a user to zoom in and out, as well as rotate the image to reveal its self-similarity, by navigating prerecorded video material. The system combines JythonMusic with Processing using OSC. For more information, see http://bit.ly/escherKinect .

3.5 Migrant (2015)

Migrant [26] is a cyclic piece combining data sonification, interactivity, and sound spatialization. It utilizes migration data collected over 23 years from 56,976 people across 545 US counties and 43 states. The piece was originally composed for *Undomesticated*, a public-art installation in the context of ArtFields 2015 (http://www.artfieldssc.org). It was performed, as part of the ISMIR 2015 music program, in Oct. 2015, in Málaga, Spain. The original is here http://bit.ly/migrant2015, and a more extended performance at the American College of Greece here http://bit.ly/migrant2016 .

3.6 SoundMorpheus (2016)

SoundMorpheus [27] is a sound spatialization and shaping interface, which allows the placement of sounds in space via arm movements. This system combines Myo armbands with JythonMusic and PureData. For more information, see http://bit.ly/soundmorpheus2 .

4. CONCLUSION

This paper provided a quick introduction to JythonMusic, a software environment for developing interactive musical experiences and systems and described some projects developed with it. Due to the limited space, only a subset of the available libraries were discussed. The online website - http://jythonmusic.org - provides additional API documentation, as well as various code samples, videos, and other references. This environment supports computer-assisted composition, image manipulation, building graphical user interfaces,

and interacting with external devices via MIDI and OSC, among other features.

5. ACKNOWLEDGMENTS

The following individuals have contributed to JythonMusic code development, API design and review, and testing: David Johnson, Paul Helling, Kyle Stewart, Margaret Marshall, William Blanchett, Christopher Benson, Mallory Rourk, Seth Stoudenmier, and Kenneth Hanson. The JEM editor is developed and maintained by Tobias Kohn. The jMusic library and materials have been developed by Andrew Brown, Andrew Sorensen, Rene Wooller, Tim Opie, Andrew Troedson and Adam Kirby. The jSyn environment is developed and maintained by Phil Burk. Chrestos Terzes provided invaluable information on ancient Greek music theory and the tetrachord. JythonMusic includes code partially supported by the US National Science Foundation (DUE-1323605, DUE-1044861, IIS-0736480, IIS-0849499 and IIS-1049554). Additional support has been provided by Google and IBM.

6. REFERENCES

[1] P. Modler. Interactive computer music systems and concepts of Gestalt. In Leman M. (eds) *Music, Gestalt, and Computing. Lecture Notes in Computer Science (Lecture Notes in Artifical Intelligence)*, vol 1317. Springer, Berlin, Heidelberg, 1997, 482-494.

[2] R. Rowe. *Interactive Music Systems: Machine Listening and Composing*. MIT Press, Cambridge, MA, 1992.

[3] G. Paine. Towards a Taxonomy of Realtime Interfaces for Electronic Music Performance. *In Proceedings of the 10th Conference on New Interfaces for Musical Expression (NIME 2010)* (Sydney, Australia, June 15-18, 2010), 436-439.

[4] E. M. von Hornbostel, and C. Sachs. Classification of Musical Instruments: Translated from the Original German by A. Baines and K. P. Wachsmann. *The Galpin Society Journal*, 14 (1961), 3-29.

[5] D. Birnbaum, R. Fiebrink, J. Malloch, and M. M. Wanderley, Towards a Dimension Space for Musical Devices. *In Proceedings of the 5th International Conference on New Interfaces for Musical Expression (NIME'05)*, (Vancouver Canada, May 26-28 2005), 192-95.

[6] G. Wang, P. R. Cook and S. Salazer, ChucK: A Strongly Timed Computer Music Language. *Computer Music Journal* 39, 4 (Winter, 2015) 10-29.

[7] J. McCartney, Rethinging the Computer Music Language: SuperCollider. *Computer Music Journal* 26, 4 (Winter, 2002), 61-68.

[8] M.S. Puckette, Pure Data: Another Integrated Computer Music Environment. In *Proceedings of the Second Intercollege Computer Music Concerts* (Tachikawa, Japan, 1996), 37-41.

[9] B. Vercoe and D. Ellis, Real-Time CSOUND: Software Synthesis with Sensing and Control. In *Proceedings of the International Computer Music Conference (ICMC)*, (Glasgow, Scotland, 1990), 209-211.

[10] J.A. Mills III, D.D. Fede and N. Brix, Music Programming in Minim. In *Proceedings of the 2010 Conference on New Interfaces for Musical Expression (NIME'10)*, (Sydney, Australia, June 15-18. 2010), 37-42.

[11] C. Roberts and J. Kuchera-Morin, Gibber: Live Coding Audio in the Browser. In *Proceedings of the 2012 International Computer Music Conference (ICMC'12)*, (Ljubljana, Slovenia, 2012), 64-69.

[12] G. Wang and P.R. Cook, The Audicle: A Context-Sensitive, On-the-fly Audio Programming Environ/mentality, In *Proceedings of the 2004 International Computer Music Conference (ICMC'04)*, (Miami, FL, USA, 2004).

[13] S. Salazar, G. Wang and P. Cook, miniAudicle and ChucK Shell: New Interfaces for ChucK Development and Performance. In *Proceedings of the 2006 International Computer Music Conference (ICMC'06)*, (New Orleans, LA, USA, 2006).

[14] S. Salazar and G. Wang, miniAudicle for iPad Touchscreen-based Music Software Programming. In *Proceedings of the 2014 International Computer Music Conference (ICMC'14)*, (Athens, Greece, Sep. 14-20, 2014), 686-691.

[15] B. Taylor, J. Allison, W. Conlin, Y. Oh and D. Holmes, Simplified Expressive Mobile Development with NexusUI, NexusUp and NexusDrop. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME'14)*, (London, UK, Jun. 30 – Jul. 4, 2014), 257-262.

[16] C. Roberts, M. Wright, J. Kuchera-Morin and T. Hollerer, Rapid Creation and Publication of Digital Musical Instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME'14)*, (London, UK, June 30-July 4, 2014), 239-242.

[17] B. Manaris and A. Brown. *Making Music with Computers: Creative Programming in Python*. Chapman & Hall/CRC Textbooks in Computing - CRC Press, 2014.

[18] B. Manaris, B. Stevens, and A. R. Brown. JythonMusic: An Environment for Teaching Algorithmic Music Composition, Dynamic Coding, and Musical Performativity. *Journal of Music, Technology & Education* 9, 1 (May 2016), 55-78.

[19] A. R. Brown. *Making Music with Java: An Introduction to Computer Music, Java Programming, and the jMusic Library*. Lulu, Raleigh, North Carolina, 2005.

[20] P. Burk. JSyn - Audio Synthesis API for Java, http://www.softsynth.com/jsyn (accessed Jan. 27, 2018).

[21] B. Manaris. Dropping CS Enrollments: or The Emperor's New Clothes?. *ACM Inroads*, 39, 4 (Dec. 2007), 6-10.

[22] P. Guo. Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities. *Communications of the ACM* (July 2014).

[23] B. Manaris, D. Hughes, Y. Vassilandonakis. Monterey Mirror: An Experiment in Interactive Music Performance Combining Evolutionary Computation and Zipf's Law. *Evolutionary Intelligence* 8, 1 (Mar. 2015), 23-35.

[24] D. Johnson, B. Manaris, Y. Vassilandonakis, and S. Stoudenmier. Kuatro: A Motion-Based Framework for Interactive Music Installations. In *Proceedings of the 40th International Computer Music Conference (ICMC'14)*, (Athens, Greece, Sep. 14-20, 2014), 2014.

[25] B. Manaris, D. Johnson, and M. Rourk. Diving into Infinity: A Motion-Based, Immersive Interface for M.C. Escher's Works. In *Proceedings of the 21st International Symposium on Electronic Art (ISEA 2015)*, (Vancouver, Canada, Aug. 14-19, 2015), 2015.

[26] B. Manaris, and S. Stoudenmier. Specter: Combining Music Information Retrieval with Sound Spatialization. In *Proceedings of the 16th International Conference on Music Information Retrieval (ISMIR 2015)*, (Málaga, Spain, Oct. 26-30 2015), 2015.

[27] C. Benson, B. Manaris, S. Stoudenmier, and T. Ward. SoundMorpheus: A Myoelectric-Sensor Based Interface for Sound Spatialization and Shaping. In *Proceedings of the 16th International Conference on New Interfaces for Musical Expression (NIME 2016)*, (Brisbane, Australia, Jul. 11-15, 2016), 2016.