JythonMusic: An Environment for Developing Interactive Music Systems

Bill Manaris, Pangur Brougham-Cook
Computer Science Dept.
College of Charleston, USA
manarib@cofc.edu, broughamcookp@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 Java, 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, sen surrogate, 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 movement, 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.
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 through 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:

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

```python
# 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](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](http://bit.ly/charlestonLaptopOrchestra).

3.2 Monterey Mirror (2011)


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](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](http://bit.ly/escherKinect).

3.5 Migrant (2015)


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/soundmorpheus](http://bit.ly/soundmorpheus).

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](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 Hellling, Kyle Stewart, Margaret Marshall, William Blanchett, Christopher Benson, Mallory Rouk, 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 Wooler, Tim Opie, Andrew Troedson and Adam Kirby. The jSyn environment is developed and maintained by Phil Burk. Chrestos Terzis 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