Multimedia-Based Visual Programming Promoting Core Competencies in IT Education

V. J. Manzo
Boyer College of Music
Temple University
Philadelphia, PA 19122
1-973-655-7212
vj@vjmanzo.com

Matthew Halper
Conservatory of Music
Kean University
Union, NJ 07083
1-908-737-4337
halper@kean.edu

Michael Halper
Information Technology Department
NJIT
Newark, NJ 07102
1-973-596-5752
michael.halper@njit.edu

ABSTRACT
Programming constitutes one of the core competencies demanded of any IT education. However, some students within certain specializations of this diverse discipline are inclined to question the need for programming. The use of a visual programming environment in the development of interactive multimedia applications can serve the dual purposes of getting students excited about programming and giving them the core knowledge they need. The visual language Max/MSP/Jitter ("Max"), geared toward music, audio, and video application programming, is introduced as an excellent vehicle toward achieving this goal. The foundational constructs of Max are introduced in a series of example programs dealing with music applications. Some details of an undergraduate IT course called "Interactive Music System Technology" that utilizes Max are presented. Overall, the use of Max in the undergraduate IT curriculum can enhance the student's experience (both in multimedia and in IT in general) and promote better programming skills.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features – *classes and objects, control structures.*

General Terms: Languages.

Keywords: IT Programming, Max/MSP/Jitter, Visual Programming Language, Multimedia, Digital Audio, Computer Music

1. INTRODUCTION
Programming—particularly the ability to do it well—is one of the pillars of IT education and a skill essential for the aspiring IT professional [1]. However, with a diverse array of specializations within this burgeoning field, some students question their need to acquire this skill. How often is it heard: "I switched to IT because I don't like to program"? Of course, students quickly learn that programming follows them everywhere in their computing pursuits, from socket programming in the networking environment to scripting in game development. And with the explosion of apps and app development, it is becoming clearer to students that this is knowledge they must obtain.

SIGITE'11, October 20–22, 2011, West Point, New York, USA.

Multimedia is a popular specialization within IT education that can lead to careers ranging from 3D artist and game designer to audio engineer and marketer. This is one area where students might especially question programming requirements. However, a vast amount of exciting work emerging from the multimedia field is custom program-based application development. In particular, a visual programming language named Max/MSP/Jitter ("Max," for short) [2,3] has gained widespread acceptance and a large user base in multimedia. Max can be utilized for many different multimedia tasks, including music, digital audio, and video. The IT educator will find it very useful in a variety of settings.

In this paper, we introduce some of the foundational aspects of Max that make it an excellent vehicle for teaching programming and data-flow to IT students. Within the IT program at the New Jersey Institute of Technology, we are piloting a course called "Interactive Music System Technology" that makes use of this language to write custom software for musical and audio interaction. While the course is primarily intended for multimedia specialists, we envision it as being of interest to all IT majors. A goal of the course is to change students' perception of programming from that of meaningless drudgery to an exciting endeavor. The students will experience first-hand what inspiring products visual programming can achieve—through their own imagination and creativity. Overall, the course represents a synergy between multimedia application development and the obtainment of the core programming competency required of the IT student. The content of this course is discussed.

2. BACKGROUND
An interactive music system is a combination of hardware and software that allows an individual to accomplish a musical/audio task, typically in real-time, through some kind of interaction. Though commonly associated with composition and performance, the tasks associated with such systems can include analysis, instruction, assessment, sound synthesis, etc. These systems have switches, keys, buttons, and sensors (in hardware and/or software) for real-time control of musical elements like harmony, rhythm, dynamics, and timbre.

Max is a programming language optimized for use in developing interactive music systems and other multimedia implementations [3]. Its extension Jitter was introduced for video processing tasks. Unlike traditional text-based programming languages, Max follows a visual programming paradigm. A Max program visually represents data-flows from one programming element (object) to another. For example, consider the audio signal chain involved in recording on your computer using a microphone and playing back via some speakers (Figure 1). Sound captured by the microphone is transmitted through a cable to the sound card, which in turn transmits the data to the computer's memory via another cable. On

the other side, cables are used to transmit the data to the speakers. In Max, each component in the figure would be denoted as an object, a programming element with a particular purpose. Data in the form of MIDI codes, audio, video, text, etc., are transmitted between objects through the use of connecting cables called patch cords in Max parlance.

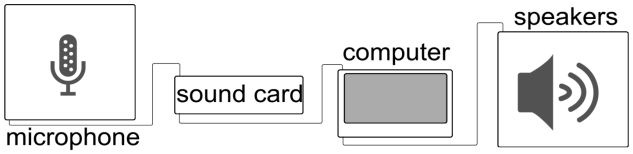

Figure 1. Audio signal flow diagram

Other data-flow environments aside from Max exist for music and multimedia applications, including Pure Data (Pd) [4], OpenMusic [5], Isadora [6], and Reaktor [7], to name a few. These environments also allow for modular programming, where reuse can be accomplished by graphically mapping the flow of data from one code module to another. We also find graphical programming development platforms in other areas. For example, LabVIEW [8] is used to build scientific measurement, test, and control systems.

MIDI is a protocol for conveying musical messages. Synthesizers as well as nearly all computer music software applications deal with MIDI in some way. In essence, MIDI is a bunch of messages in which 8-bit numbers (encapsulating 7-bit data representations, i.e., 0–127) are used to represent musical elements like pitch and velocity. The lowest MIDI note 0 is the pitch C at five octaves below middle C. The number 1 is the C#/Db directly above that C, and so on. The MIDI note 12 is the C an octave (twelve half-steps) above the lowest C. Velocity is a measure usually associated with (note) volume represented by the same numbers 0–127. The value 0 usually means that the note is silent, whereas the value 127 is the loudest volume. Therefore, the note 60 with a velocity of 127 could be described as middle C at a very loud volume.

3. PROGRAMMING CONSTRUCTS OF MAX

The Max programming environment is really just a canvas on which objects are organized into a model of data-flow. Data is transmitted from object to object through the patch cords at run time. A collection of interconnected objects and associated patch cords is called a *patch* (really, a visual program). Graphically, each object is represented as a circle or round-edged rectangle, and a patch cord is a thin black connecting line. At the top of each object, we find one or more *inlets* that accept input from other objects via patch cords. At the bottom of objects are *outlets* that allow for subsequent output along other patch cords. Max includes an extensive built-in library of classes from which a large variety of objects can be instantiated. These objects range from simple numbers and text to complex multimedia entities such as audio signals and visual matrices. Many of these objects model the standard control structures of a high-level programming language. (Let us note that Max does not use the term "class" to describe this arrangement as found in traditional object-oriented programming languages such as C++ and Java.) In the following, we present some of the fundamental constructs of Max through a series of example patches dealing with creating random music.

A small sample patch consisting of three objects and two patch cords can be seen in Figure 2. The middle object called *random* is shown with the added argument 128. It outputs a random number between 0 and 127 when it receives an event message in its left inlet. The small circular object above *random* is a *button* object, a user interface component that sends out event messages called "bangs" when a user clicks on it. The *number* object shown beneath *random* simply serves to display the numerical value generated by *random* when a user requests it via a click.

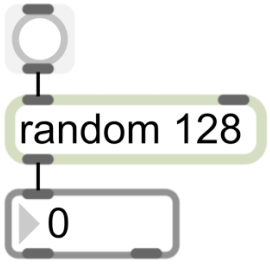

Figure 2. Max patch to output a random number (0–127)

With the Max objects in Figure 2, we are able to generate random numbers that can be used as MIDI pitches. We can wrap these into MIDI messages along with the velocity information and note duration using the MAX object *makenote* (Figure 3). This object takes two numbers as arguments to specify the default velocity (first argument) and default duration in milliseconds (second argument) that will be associated with the random pitch value. The note will last until *makenote* sends out a velocity 0 value to turn it off, as indicated by the bottom-right object in Figure 3. Overall, the patch in Figure 3 will produce a random MIDI note (with velocity 100 and duration 500 milliseconds) on demand with a click of the button at the top. This patch nicely demonstrates the fundamental constructs of input/output, data structuring, and time management within the Max framework.

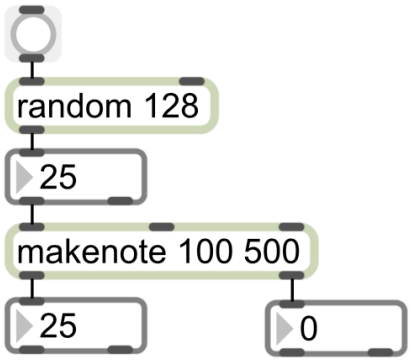

Figure 3. Format a MIDI message using a random number as pitch

At the moment, the patch in Figure 3 simply displays the random pitch (bottom left object). Of course, in the context of music system development, it is much more appropriate to have the note actually played. This can be done using the *noteout* object, as seen in Figure 4, which sends the formatted MIDI message to the computer's sound card where it is synthesized into an actual note.

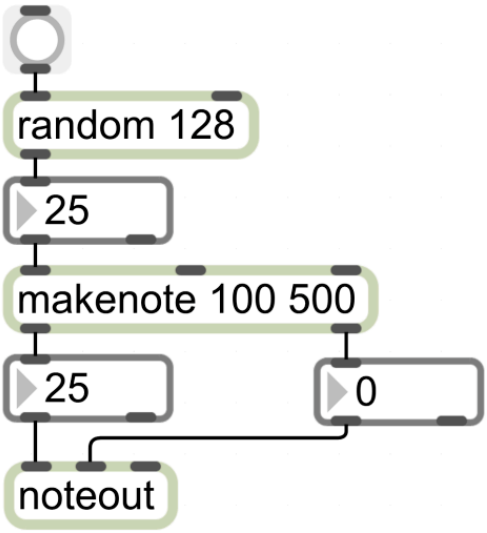

Figure 4. Send a random note to the sound card

As an extension, let us look at a patch that automatically generates a new random note each half second. This can be managed with the use of a *metro* object (Figure 5) that acts like a metronome and sends out "bangs" at a specified interval of time. The *metro* in Figure 5 bangs at an interval of 500 milliseconds.

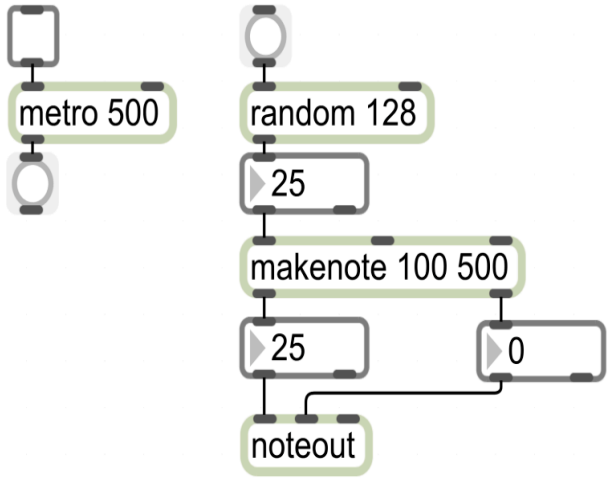

Figure 5. Metronome object to output a "bang" every 500 milliseconds (not connected to other code)

The square above the *metro* object is another user interface object called a *toggle*, which, when clicked by the user, turns *metro* on or off. Patching together the random note generator of Figure 4 with the *metro* and *toggle* combination of Figure 5 gives us our desired patch, shown in Figure 6.

When the user clicks on the single *toggle* object, random pitches are played every half second. When the toggle is clicked off, the pitch creation ceases.

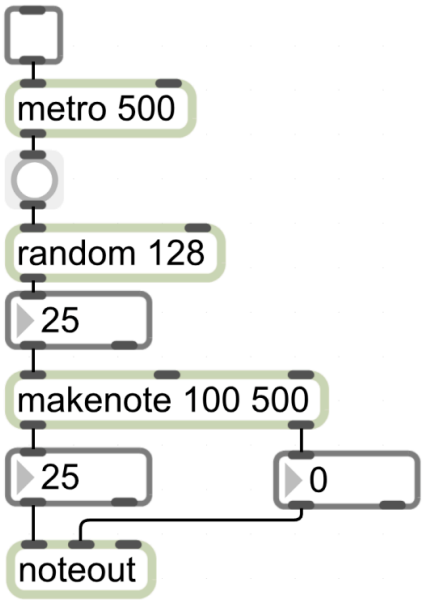

Figure 6. Toggled random pitch playback

Supplying arguments to objects via additional *number* objects allows users to manipulate arguments in real-time. In Figure 7, *number* objects (with current values of 300) connected to *metro* and *makenote* control, respectively, the speed and duration of generated pitches. A user can change these simply by typing in new numbers. Let us point out that the numerical argument value, serving as the default (e.g., 500 within *metro*), will not change visually within the object's appearance in the patch. However, that value will not be used as long as a positive number is fed into its inlet. The default value of 500 will again be used when the patch is closed and later reopened (re-run).

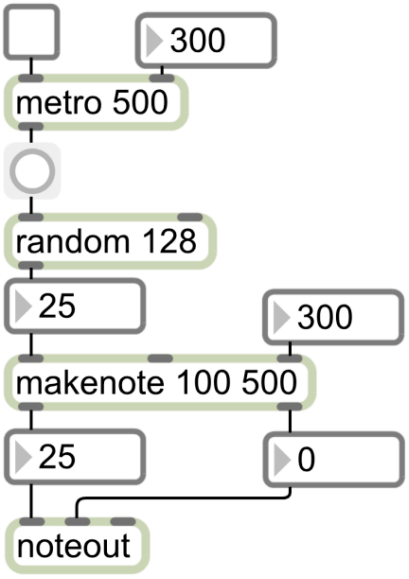

Figure 7. Changing default arguments of *metro* and *makenote*

More intuitive graphical control can be obtained with the use of a *slider* object patched into the two *number* objects (Figure 8). A single *slider* is used to keep the arguments in sync. Separate sliders could be used for each value.

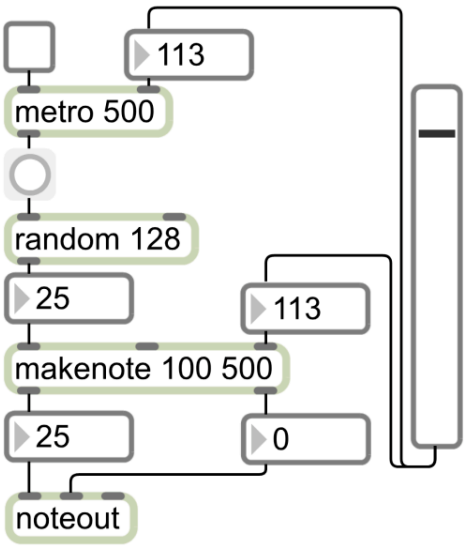

Figure 8. *slider* control of argument values

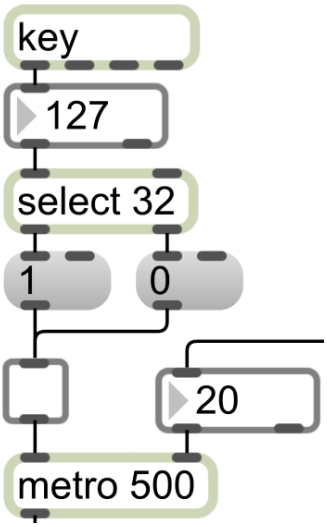

Figure 9. On/off toggle based on the spacebar

As an extension to demonstrate a programmatic selection (*if-else*) structure, consider the partial patch shown in Figure 9. The *key* object gets the ASCII key number on a key press by the user and outputs it. For example, when the user hits the spacebar (ASCII value 32), a value of 32 emerges. (In this patch, the value is displayed by the *number* object beneath *key*, though that is not required.) The output of *key* is delivered to the *select* object, which performs a comparison with the value 32. If the input value = 32, the *select* will send a bang from its left outlet triggering a *message box* containing the number 1 to send that value 1 to the *toggle* to turn it on. Otherwise, *select* will send a bang from its right outlet triggering a message box with 0 to send out that value to the toggle in order to turn it off. (Note that this is a slightly different use of the toggle from that used above; it is being controlled directly by other objects rather than being clicked by the user.) As we see, *if-else* is implemented as an object in Max. Interestingly, the results of the selection are alternate "bangs" that emanate from the object. Thus, the programmer can get real tangible feedback in the formulation and execution of such a foundational programming construct. More elaborate selection statements encompassing more extensive conditions and clauses can be built with this and other objects in Max including *expression* and *if* objects.

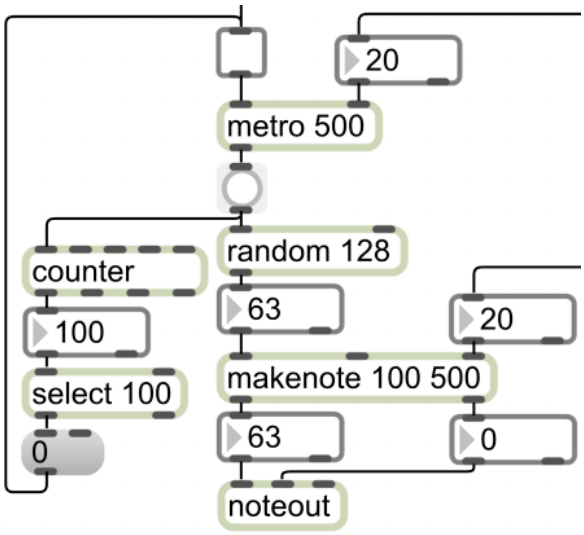

Figure 10. *counter* object detects 100 bangs from *metro* and stops the loop

Max, of course, accommodates looping, as demonstrated by the patch in Figure 10. This patch generates 100 random notes and then terminates. This is achieved by the use of a *counter* object that counts the number of bangs sent from *metro* to *random*. When *counter* reaches 100, the *select* object beneath *counter* will cause the message box containing 0 to send the value 0 to the toggle above *metro*, thus turning it off. Let us point out that this random-note loop is initiated by a press of the spacebar as in Figure 9.

4. AN IT UNDERGRADUATE COURSE BASED ON MAX

Max is being used as the development vehicle for a course titled "Interactive Music System Technology" that is being piloted in the Information Technology Department at the New Jersey Institute of Technology. At present, the course only requires that the student has taken the introductory IT sequence, which affords them some familiarity with a high-level programming language. However, knowledge of high-level programming is not mandatory for this course. No music background is assumed, though it is beneficial. IT aspects of the music/audio domain are emphasized throughout the course.

As noted, one of the primary goals is to get students comfortable with, and indeed excited about, programming as the means to achieving interesting and important results in IT. In particular, the students learn how to write custom software to create interactive music systems. Substantial projects based on Max are required of all class members. Foundational topics include digital music/audio protocols and representations (e.g., MIDI and OSC), synthesis, sequencing, and signal processing. Beyond that, emphasis is placed

on the development of interactive software for music creation and processing using algorithmic techniques in the context of Max. The many applications of interactive music software, such as music composition and performance, live audio installations, creation of new musical instruments, music analysis, scoring for film and multimedia, and instructive and assistive technologies, are covered.

The specialized IT hardware used to support the interactive software is considered. Interaction techniques with devices beyond the mouse and keyboard, such as the use of camera tracking, pitch tracking, video game controllers, sensors, and mobile devices, are explored. Some of Max's video processing is also dealt with in the course.

A laboratory of iMacs equipped with external MIDI keyboards is being used for the course. In addition to Max, software synthesizers and digital audio workstation applications are incorporated to introduce interactive performance and composition.

5. DISCUSSION

Max is a powerful visual programming language for IT multimedia specialists looking to design customized software for audio and music (as well as video) applications. It offers a very intuitive programming paradigm that reflects the flow of data from one structure to another. In fact, everything in Max is implemented as an object, which, in addition to graphically rendering basic programming constructs, promotes object-oriented thinking and object-oriented approaches to problem solving. As such, Max is an excellent vehicle for teaching programming skills and providing IT students with the core competency in programming they need. Beyond the multimedia specialist, a course based on Max would benefit all IT students, and perhaps computing students in general. Moreover, since Max/MSP/Jitter is optimized for music and video applications, students using it will be able to create exciting applications right from the start, instead of having to deal with the classic data crunching projects often assigned in the fundamental programming courses. As an additional benefit, the Max environment supports rapid prototyping and can give students the satisfaction of immediate feedback on their work.

Although Max is robust, the Max SDK allows users to write their own objects in C++ to extend the language. In addition, there is support for JavaScript within the environment. Third-party objects have also been written to encapsulate other languages such as Lisp and CSound within Max [3]. This flexibility offers IT students further opportunities to enhance their programming skills by adding their own customized features to the language.

Max has been deployed in a wide variety of multimedia applications. EAMIR [9] is an interactive music system and open-source software project that allows individuals to create music without the physical and technical limitations found in performing on traditional musical instruments. Software is designed in Max to allow individuals to create music by using physical gestures or interfaces such as game controllers (Wii remotes, Guitar Hero controllers, DDR pads, etc.), cameras, sensors, and more. IMTCP [10] was a research project for teaching music to students who had no prior musical knowledge by using software-based musical instruments. The Modal Object Library [11] is an open-source collection of third-party objects written to extend Max. The library provides tools for filtering random numbers to notes from selected musical scales and provides a framework for chord creation, algorithmic composition, interactive music installations, and performance. Other Max projects include composition and performance [12], video game integration, education, and research (see, e.g., [13]), and musical instrument design [14]. All these application areas are fertile ground for student projects within the purview of an IT class or, for example, for capstone projects in the overall IT curriculum.

Of course, the issue of teaching introductory programming, particularly object-oriented programming, has been the focus of much research over the years and has been written about extensively in various forums, e.g., in the context of the ACM's SIGCSE. And a variety tools have been developed for the express purpose of teaching programming. BlueJ [15,16] is a popular example whose approach centers on the visualization of classes and objects and allows for interactive manipulation of objects by students without the need for constructing cumbersome "driver" apparatuses. However, BlueJ is expressly not a vehicle for GUI building and is not concerned with multimedia applications *per se*. The "MediaComp" approach [17] uses the manipulation of multimedia elements to inspire students (both computing and non-computing) to learn programming. For example, loops in Python [18] are deployed in the service of modifying pictures and sound. The Max language, on the other hand, is a full-blown visual language for building multimedia applications, with a built-in suite of objects for representing and manipulating music, sound, and video. From the examples of its use described herein, Max can be seen as another component in the repertoire of pedagogical programming tools, one that is particularly well-suited for the IT student as opposed to the traditional computer science student.

Admittedly, studies would need to be conducted to determine the actual efficacy of Max in motivating students who would not otherwise be inclined to program to pursue this endeavor with enthusiasm. Moreover, it would be appropriate to analyze the effectiveness of Max in promoting students' fundamental understanding of programming constructs and to compare the results with other approaches. But, in the context of our piloted course, the goal is not just programming but interactive multimedia system development, and Max—a professional development language—has been well tested in this regard by an extensive user community.

6. CONCLUSION

Programming is one of the core competencies of any undergraduate IT student. The Max visual programming language, used widely in the digital music/audio field, has been proposed as an excellent platform from which to provide IT students with this requisite skill. The fundamental constructs of the Max language have been presented, and an IT course, "Interactive Music System Technology," using Max as the vehicle for software development has been discussed. Overall, the Max visual programming environment and its focus on multimedia applications promises to make IT students excited about the potential that programming holds and encourage them to pursue it more vigorously. While such a course has been proposed as a core component of a multimedia specialization in IT, we see it as enhancing any undergraduate IT curriculum.

7. REFERENCES

[1] B. M. Lunt, J. J. Ekstrom, Sandra Gorka*, et al.* Information Technology 2008: Curriculum Guidelines for Undergraduate Degree Programs in Information Technology, ACM and IEEE Computer Society, November 2008, available at http://www.acm.org/education/curricula-recommendations. Accessed May 3, 2011.

[2] Max - Cycling 74, available at http://cycling74.com/products/maxmspjitter. Accessed May 12, 2011.

[3] V. J. Manzo, Max/MSP/Jitter for Music, Oxford University Press, New York, 2011.

[4] Pure Data - Pd Community Site, available at http://puredata.info. Accessed May 25, 2011.

[5] OpenMusic, available at http://repmus.ircam.fr/openmusic/home. Accessed May 27, 2011.

[6] TroikaTronix - Isadora, available at http://www.troikatronix.com/isadora.html. Accessed May 27, 2011.

[7] Reaktor, available at http://reaktor.en.softonic.com. Accessed May 27, 2011.

[8] NI LabVIEW, available at http://www.ni.com/labview. Accessed July 30, 2011.

[9] EAMIR, available at http://www.eamir.net/forum. Accessed May 10, 2011.

[10] Interactive Music Technology Curriculum Project (IMTCP), available at http://www.imtcp.org. Accessed April 29, 2011.

[11] Modal Object Library, available at http://www.vjmanzo.com/mol. Accessed May 25, 2011.

[12] R. L. Dubois. Applications of Generative String-Substitution Systems in Computer Music. Doctoral dissertation, Columbia University, New York, NY, 2003.

[13] Adaptive Use Instruments Project. Deep Listening Institute, available at http://deeplistening.org/site/adaptiveuse. Accessed May 30, 2011.

[14] A. Chowdhury, S. Cho, and U. Chong. Musical controller for wind instrument using Max/MSP software and ATmega128. J. Acoustical Society of America, 129(4): 2543, 2011.

[15] BlueJ - The Interactive Java Environment, available at http://bluej.org. Accessed August 9, 2011.

[16] D. J. Barnes and M. Kölling, Objects First with Java: A Practical Introduction using BlueJ, 4th Edition, Pearson, Upper Saddle River, NJ, 2008.

[17] Media Computation Teachers Website, available at http://MediaComputation.org. Accessed August 9, 2011.

[18] M. Guzdial and B. Ericson, Introduction to Computing and Programming in Python: A Multimedia Approach, 2nd Edition, Pearson, Upper Saddle River, NJ, 2010.