Pocket Gamelan: a Pure Data interface for mobile phones

Greg Schiemer
Faculty of Creative Arts
University of Wollongong
Wollongong NSW Australia
61 2 4221 3584
schiemer@uow.edu.au

Mark Havryliv
Faculty of Creative Arts
University of Wollongong
Wollongong NSW Australia
61 2 4221 3584
mhavryliv@hotmail.com

ABSTRACT

This paper describes software tools used to create java applications for performing music using mobile phones. The tools provide a means for composers working in the Pure Data composition environment to design and audition performances using ensembles of mobile phones. These tools were developed as part of a larger project motivated by the desire to allow large groups of non-expert players to perform music based on just intonation using ubiquitous technology. The paper discusses the process that replicates a Pure Data patch so that it will operate within the hardware and software constraints of the Java 2 Micro Edition. It also describes development of objects that will enable mobile phone performances to be simulated accurately in PD and to audition microtonal tuning implemented using MIDI in the j2me environment. These tools eliminate the need for composers to compose for mobile phones by writing java code. In a single desktop application, they offer the composer the flexibility to write music for multiple phones.

Keywords

Java 2 Micro Edition; j2me; Pure Data; PD; Real-Time Media Performance; Just Intonation.

1. INTRODUCTION

In the past decade there has been a paradigm shift from desktop to ubiquitous computing. Mobile phones represent a major part of this shift. The tools described here have been developed to address the challenges of composing music for such a computing environment. They were developed as part of a project called the Pocket Gamelan.

1.1 Pocket Gamelan - background

The Pocket Gamelan project seeks to develop a software prototype for an interactive musical performance interface that can be used by non-expert performers [1,2]. Using mobile phones as the basis for this interface, the project seeks to explore new tuning systems and enable performance by large numbers of non-expert performers playing music based on just intonation using hand-held or wearable instruments [3].

In previous papers we described how performance scenarios associated with the project might extend the musical legacy of historical tuning systems as well as new tuning systems first explored by composer and theorist Harry Partch [4] and extended through the work of contemporary tuning theorist Erv Wilson [5,6,7,8,9].

An application based on one of these performance scenarios has been implemented as a library of j2me classes [10,11,12,13,14]. A Nokia 6230 mobile phone was used as the target device. A new work based on this tuning has been composed. Entitled *Mandala 3* it will be performed at UK Microfest, Riverhouse, Walton-on-Thames, London, UK, October 15th, 2005. The composition is based on one of Wilson's product set tunings called the Euler-Fokker Genus [15]. In performance each mobile phone is swung on the end of a cord. This produces audible artifacts such as Doppler shift as a bi-product of movement. The performance scenario originated from mobile instruments developed by one of the authors two decades earlier [16,17].

On this occasion, four mobile phones will be used. In the lead up to this performance, software development has concentrated on two features of j2me. They include: real-time audio generation and a microtonal MIDI implementation.

Up to this point, the Pocket Gamelan project has already encountered many of the initial obstacles to achieving real-time performance using a mobile phone. The most common of these are the limited resources of j2me devices and irregular media implementations. A more accessible interface to compose music for mobile phones has been developed using a new application called pd2j2me.

1.2 pd2j2me

Pd2j2me is a desktop Java application that cross-compiles from PD to j2me. It allows musical applications composed for performance on mobile phones to be simulated using Pure Data.

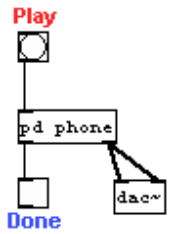

Figure 1. PD Patch shows PD object called phone. One inlet (Play) starts a process that simulates the j2me application. Left outlet (Done) flags the state of the process while the right outlet allows the object to produce sound.

A j2me device is represented as a PD patch with one inlet and two outlets as shown in Figure 1. The pd2j2me compiler creates a source file that can be exported to the j2me environment.

A composer may create a PD patch with many such phone objects in order to simulate the realisation of a musical performance using multiple mobile phones.

Development of pd2j2me is driven by the requirements of performance scenarios associated with the Pocket Gamelan project where musical applications were initially written in j2me. An understanding of the limitations of mobile phone technology acquired in the process of creating these applications has informed our decision to build a cross-compiler rather than a run-time PD interpreter for j2me.

2. Design Considerations

A j2me version of PD is not a feasible option for time-critical operations like audio, MIDI and user interaction which require continual optimisation of resources. To overcome this problem, a compiler is used to generate j2me source code that will emulate the behaviour of a program written as a PD patch. It is then possible to download j2me code into the mobile handset and execute it.

2.1 Compilation

The compiler creates a j2me source file consisting of a hybrid collection of:

- simple j2me expressions such as *var* = *arg* + *const*;
- calls to purpose-built j2me classes that support more complex objects such as line, metro, delay, select, counter and array; and
- interfaces to purpose-built classes that enable system IO like audio, access to visual output and user input, and Bluetooth wireless connection.

2.2 Simple Objects

Simple objects are classed as those that can be contained in one line of j2me code and all of whose arguments can be accessed in local or global variable space. These include simple arithmetic, tabreads, sends, receives and messages.

Simple objects are written to accommodate the possibility of changing an argument after it has been initialised. This is done by creating a global variable. This is stored as a second argument that may be changed by other processes at run time.

```
public class SampleProg {
double mv0, mv1, minus1v1, plus3v1;
public SampleProg() {
                mv0 = 0;
                mv1 = 0;
                minus1v1 = 7.0;
                plus3v1 = 3.0;
                PDMain();
        }
```

Figure 2. *SampleProg()* initialises a value for objects *minus1v1* and *plus3v1* whose arguments may be altered during program execution.

2.3 Complex Objects

Some PD's objects cannot be contained in one line of j2me code. To overcome this problem, a small library of general purpose objects were created to simulate the behaviour of more than one PD object.

For example, a single timer-based object in j2me can simulate the behaviour of a metro, delay, line, line~ and vline~ object just by changing the way it is called at run-time. This ensures that the fastest possible algorithm is used for the core operation.

Values returned by complex objects in the j2me environment are represented as separate methods in the PD program. All possible execution threads leading from the object are called from that method.

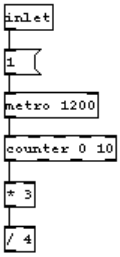

Figure 3. A PD patch demonstrating two complex objects metro and counter. Metro assumes control over program flow.

```
public void met0Bang() {
        double a;
        a = counter.bang();
        a = a * 3;
        a = a / 4;
}
```

Figure 4. Example shows complex object *counter* and simple objects compiled to j2me where *met0Bang()* is called whenever a metro event occurs whereupon *counter* returns an incremented value. *met0Bang()* is run as a separate thread without interrupting other execution streams.

2.4 Program Flow

Program flow is first visualised as streams of data then replicated in j2me. Each phone object can have a number of inlets. One of these inlets must have a toggle or a bang connected to it. Other inlets are assumed to be wireless connections from other phones. A bang or a toggle on one inlet is necessary to start the phone application. From this inlet the compiler determines the execution order for all code within the phone.

A stream begins when:

- an inlet object receives a toggle or a bang; or
- an object is connected to more than one other object.

A stream ends when either:

- a series of patch cords is terminated by connection to a non-live inlet;
- an object's outlet is connected to more than one inlet;
- a multiple output object like trigger or select is used; or
- an object's outlet is not connected to anything.

J2me program flow follows the connection order associated with visible objects used in a PD patch. This can either be implicit - which cannot not be illustrated graphically - or explicit as in the case of PD objects with multiple outputs.
An example of this is the trigger (or t) object where multiple outputs are triggered from right to left. The t object shown in

figure 5 has 4 float outputs. Because these are sequenced from right to left, the t object provides a visible means to determine the order in which j2me methods are executed.

Using the PD patch shown in figure 5, pd2j2me compiles j2me source code shown in figures 2, 6 and 7. The PD patch shown in figure 5 is a representative collection of possible ways that program flow may be altered in PD; output code examples show how these are represented in j2me.

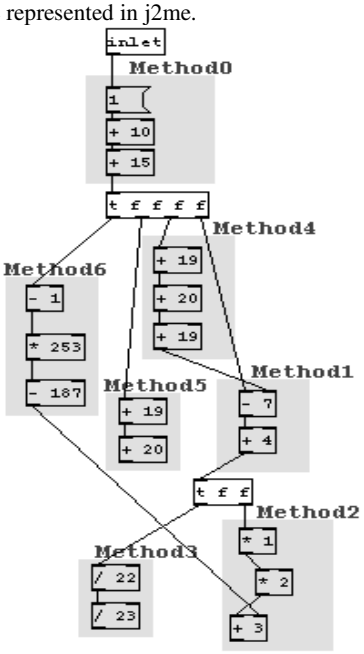

Figure 5. A PD patch containing all likely combinations of program flow and variable assignment.

Each PD data stream is represented by a method and may be passed or may return a value. *PDMain()* , shown in figure 6, is a method run immediately after initialisation. It sequences seven methods in their order of execution.

```
public void PDMain() {
        mv0 = method0();
        mv1 = method1(mv0);
        method2(mv1);
        method3(mv1);
        minus1v1 = method4(mv0);
        method5(mv0);
        plus3v1 = method6(mv0);
}
```

Figure 6. Execution begins in PDMain(). Data streams shown in Figure 5 are represented by 1 of 7 methods shown here.

2.5 J2ME Object Creation

In *method0()* and *method1()*, only one local variable is created. Objects are created only if they are required to replicate the behaviour of the PD patch. This reduces the frequency at which garbage is collected by the j2me virtual machine.

```
public double method0() {
        double a;
        a = 1;
        a = a + 10.0;
        a = a + 15.0;
        return a;
}
public double method1(double arg0) {
        double a;
        a = arg0 - minus1v1;
        a = a + 4.0;
        return a;
}
```

Figure 7. *method0()* (up to the t object) is executed first. *Method1()* starts at the – 7 object and receives *arg0*, an argument from *method0()*. *Arg0* is the left operand of –7 while *minus1v1 is* the right variable whose value may be altered by program events.

2.6 System Interfaces

The set of classes described above, enable real-time audio and microtonal MIDI output from a mobile phone. These classes provide a robust interface to handle audio and MIDI effectively. A similar set of classes were developed to access the mobile phone screen output and its user input. Regulation of access to these system resources asserts a level of runtime integrity not possible if resources are allocated as soon as they are requested.

2.7 Wireless Communications

A stand-alone Bluetooth library has also been written to manage wireless communications between mobile devices. This library will replicate control between devices as simulated in the PD patch. Bluetooth communication paths will be derived from connections made between phone objects in PD like those shown in figure 8.

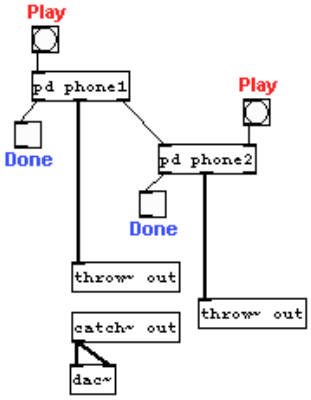

Figure 8. A representation of bluetooth connection between two mobile phones in PD. The right-most outlet of phone1 is connected to the left most inlet of phone2 and demonstrates a one way wireless communication path. Throw~ and catch~ may also be used instead of multiple dac~ objects.

3. PD User Considerations

A pd phone object is a sub-patch where a composer is expected to place composition algorithms. In order for pd2j2me to simulate a mobile phone performance, users must adopt the convention of placing all algorithmic composition code in these sub-patches. A further PD object is planned: detuned_noteout. Together these objects will be used to simulate microtonal performance behaviours envisaged in the Pocket Gamelan project.

3.1 pd phone

The phone object allows simulation of these and other behaviours using PD as a simulation environment. Pd phone encapsulates the behaviour of a mobile phone and its relationship to the real world through audio, wireless communication, screen output and user

input. The pd phone object is a sub-patch and any code inside that patch will be compiled for the real device.

A pd phone object has one inlet that expects a bang to start the application. The object's outlets represent the screen output on the phone. Any visual object like a bang, toggle or print to which a phone outlet is connected will appear on the screen, sometimes with an optional label. Because the screen size of phones vary, at the moment it is more practical to allow the compiler to decide where objects should be placed on the screen. Non-visual objects will be ignored unless they are produce audio (e.g. like dac~ or noteout) or wireless communications which will be managed by connecting phones to each other through a second inlet.

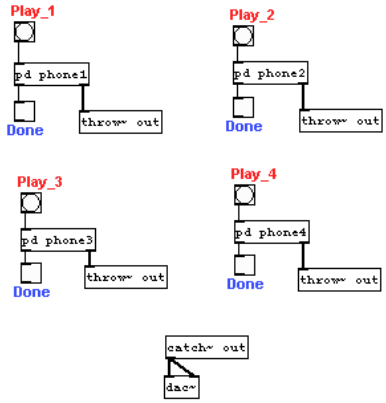

Figure 9. Example of the functionality of the phone object. The bang connected to the inlet of each phone starts the application and the toggle from the left outlet will be printed on the screen of the mobile phone.

3.2 detuned_noteout

The second object will be a microtonal noteout. This object will accept a floating point number as an argument, determine the correct pitch bend and output the detuned note. This will be organised in PD in a way that is consistent with microtonal MIDI as it is implemented in the Nokia 6230 mobile phone.

When a floating point value is passed to noteout, the object selects a MIDI channel, transmits the floating point value as a MIDI pitch bend and then performs the note. This replicates the noteout system designed for microtonal performance in j2me devices by dynamically allocating channels to manage up to 16 note polyphony.

4. Conclusion

The toolset described above offers an interface that will allow composers familiar with PD or PD-like programs such as MaxMSP to develop new musical applications for mobile phone handsets and do it quickly. It also opens the way for new communities of musicians to become engaged with creative uses of this technology so that its creative development is driven by diverse user interest groups rather than corporate cartels.

5. ACKNOWLEDGMENTS

This project was funded by an Australian Research Council Discovery Grant for 2003-2005.

6. REFERENCES

[1] Schiemer, G., Sabir, K. and Havryliv, M. 2004 "The Pocket Gamelan: A j2me Environment for Just Intonation" Proceedings of ICMC2004 University of Miami, Florida, November 4th-9th pp. 654-657

[2] Schiemer, G. "Pocket Gamelan: building the instrumentarium for an extended harmonic universe", Proceedings of International Computer Music Conference ICMC'04 Boundaryless Music, National University of Singapore, International Computer Music Association, San Francisco pp. 329 -332.

[3] Schiemer, G. and Havryliv, M. 2004 "Wearable Firmware: The Singing Jacket" Proceedings of ACMA'2004 University of Victoria, Wellington, July 1st-3rd pp. 66-71

[4] Partch, H. 1949 Genesis of Music Da Capo Press

[5] Wilson, E. 1961 Musical Instrument US Patent Office Patent Number 3,012,460

[6] Wilson, E. 1967 Musical Instrument Keyboard US Patent Office Patent Number 3,342,094

[7] Wilson, E. 1975 (1)"The development of Intonational Systems by Extended Linear Mapping"

[8] Wilson, E. 1975 (2) "Bosanquet – A Bridge – A Doorway to Dialog" Xenharmonikôn 3 (13 pages)

[9] Wilson, E. 1986 "D'Alessandro Like a Hurricane" Xenharmonikôn 9 pp. 1-38

[10] Sun Microsystems, 1999 Java 2 Micro Edition

[11] Connected Limited Device Configuration (CLDC) 1.0, Mobile Information Device Profile (MIDP) 1.0.

[12] http://www.devx.com/Java/Article/21850/0 - Optimizing Fixed Point Math with j2me. André de Leiradella 2004.

[13] http://www.javaworld.com/javaworld/jw-03-2001/jw-0309-games.html - j2me: The next major games platform? Jason R. Briggs 2001.

[14] http://developers.sun.com/techtopics/mobility/midp/ttips/optimize/ - j2me Optimization Tips and Tools. Eric D. Larson 2002

[15] Op de Coul, M. 1992 Scales Archive in Scala available at http://www.xs4all.nl/~huygensf/scala/

[16] Schiemer, G. 1999 "Improvising Machines: Spectral Dance and Token Objects" Leonardo Music Journal 9 MIT Press pp. 107-114

[17] Atherton, M. 1991 Australian Made - Australian Played University of New South Wales Press, Sydney pp. 210-211

[18] Bischoff J., Gold R. & Horton J. 1978: "Music for an Interactive Network of Computers" Computer Music Journal 2 (3) pp. 24-29