2005

Pocket Gamelan: an extensible set of microtonal instruments

Greg Schiemer
University of Wollongong, schiemer@uow.edu.au

Mark Havryliv mh675@uow.edu.au
University of Wollongong

Publication Details

This conference paper was originally published as Schiemer, G and Havryliv, M, Pocket Gamelan: An Extensible set of microtonal instruments, in Opie, T & Brown, A (eds), ACMC05 Generate and Test : Proceedings of the Australasian Computer Music Conference, Australasian Computer Music Association, 12 - 14 July 2005, 128-131.

Pocket Gamelan: An Extensible Set of Microtonal Instruments

Greg Schiemer

and Mark Havryliv
Faculty of Creative Arts,
University of Wollongong
Wollongong, 2522
Australia
schiemer@uow.edu.au
mhavryliv@hotmail.com

Abstract

This paper describes the prototype for a set of mobile instruments in which java phone technology has been adapted for performing microtonal music. The prototype was developed using widely available mobile phone handsets instead of building new hardware. The paper discusses aspects of j2me development together with limitations of the mobile platform used for the project. Development issues such as real-time audio, microtonal MIDI implementation and control using Bluetooth communication are discussed. The paper also describes tools developed so existing algorithmic composition and tuning software can be used to compose music for mobile devices. It concludes with discussion of various performance scenarios for mobile electronic instruments to realise music composed in tuning systems outside twelve-note divisions of the octave.

Introduction

The Pocket Gamelan is an ensemble of mobile musical instruments that are designed to be easy to play, quick to learn and produce audible tones that are microtonally tunable [Schiemer et al 2004]. The musical ensemble consists of large numbers of mobile phone handsets each operated independently by a single user. Each unit functions either as a control device, a sound source or some combination of both. Each unit is battery powered and able to take advantage of new developments in mobile digital computing [Schiemer et al 2003].

The prototype has been developed in java using mobile technology known as Java 2 Micro Edition, or j2me [Java Community Process 2003, 2002a]. J2me is used in hand-held appliances such as palm pilots and mobile phones.

The Pocket Gamelan project seeks to develop a software prototype for a networked ensemble of hand-held mobile instruments. Wireless communication between hand-held instruments allows enhanced interaction between ensemble members.

In its simplest form, each unit in the ensemble is a hand-held sound source that is played by pressing buttons. Players have the freedom to move each sound source while performing.

Wireless communication will extend the capabilities of the mobile user-interface by allowing button controls to be operated remotely. These capabilities will be further enhanced as wearable wireless sensors are included. In the long term, a wireless network will allow musical applications to be distributed between a web-server and multiple clients.

The aims of the Pocket Gamelan project are:

- to create a prototype network of mobile instruments for performing music free of the tuning constraints associated with conventional music performance interfaces;
- to use this prototype to explore current developments in tuning theory using new performance paradigms.

The difference between music created using this technology and music created using conventional electronic music systems such as MIDI hardware and desktop software is the degree of mobility and autonomy that a mobile instrument gives to each player. The extent to which this affects musical outcomes is limited only by the ways in which performers are allowed to move and the kinds of spaces where performances are presented.

Whereas desktop computing tends to concentrate the means of producing music in the hands of a single user, mobility offered by this technology introduces new possibilities for musical interaction between members of an ensemble. 'Gamelan', in the title, is a musical metaphor for this kind of group interaction.

Performance scenarios

Three performance scenarios have been identified.

Scenario 1: audio-yo

In scenario one multiple sound sources are fine-tuned by performers who select 'preferred intervals' as part of the performance process. The scenario is based on a work created by the principal author in 1981 [Schiemer et al 2004] in which an ensemble of battery-operated mobile sound sources are swung in a circular motion on a 1-metre cord to produce doppler shifted variants of pre-set pitches.

Scenario 2: group pokemon

In scenario two ensembles of performers trigger pre-tuned MIDI note events or activate sequences of pre-tuned MIDI note events. These are locally activated by pressing buttons on individual handsets.

Scenario 3: remote interaction

The third scenario is an extension of the first two scenarios except that one performer may actively modify sounds created by another. Buttons pressed locally activate commands that affect other clients in the network. By operating a single key one may modify the pitch or amplitude envelope or a tuning algorithm running on a remote device.

Development Environment

Musical applications for mobile devices were initially developed in j2me using a java desktop development environment in Windows XP.

Program code is first developed on the desktop using IBM's project development environment called Eclipse. Sun's Java Wireless Toolkit is used to simulate the operation of the handset. Once applications are tested and simulated on the desktop a Nokia 6230 mobile phone handset is then used as the target device. Prior to performance, applications consisting of java code are downloaded into the handset using a USB cable.

In effect the mobile handset becomes a generic hardware platform for software musical instrument applications which define the musical functionality of the handset. It is possible to implement new performance scenarios by developing new software instruments that redefine the functionality of each mobile handset.

The Nokia 6230 was chosen as the j2me target device because it was one of the first commercially available java phones to implement the Connected Limited Device Configuration protocol CLDC1.1 [Java Community Process 2003]. Though this has proved not to be a critical issue for implementing alternative tuning systems, the quality of its sound together with its support for the MIDI Tuning specification made this phone a satisfactory choice for the initial stages of the project.

j2me

j2me development has focused on three areas:

- Real-time audio generation
- Microtonal MIDI implementation
- Bluetooth control

Two of these - real-time audio generation and microtonal MIDI implementation - form the framework for managing the allocation of media resources within a compiled j2me application.

Real-time audio generation

In order to support scenario 1, an applet was developed to generate multiple sine-tones. Tuning was implemented using the variable sampling increment technique with interpolation; for more detail see [Schiemer et al 2004, 2003].

However, as the Nokia 6230 does not support synthesis of audio in real-time, this applet has not yet been tested on a handset. We are still looking for other suitable j2me target devices that support streaming audio as this will allow wave-table synthesis, additive synthesis, frequency modulation and formant wave synthesis.

Microtonal MIDI implementation

The Nokia 6230 supports MIDI files. However, its support for raw MIDI does not fully comply with the MIDI standard. While this made it unsuitable for scenario 2, it was possible to use it for scenario 1. An oscillator was made by sending a MIDI Note On followed by streams of MIDI channel messages to control a sound envelope. MIDI Controller 7 messages produce continuous amplitude envelopes. A single Note On and MIDI Pitch Wheel messages, together with MIDI portamento, produce continuous pitch envelopes.

Despite MIDI tuning resolution in the Nokia 6230, this implementation of microtonal MIDI is a workaround. We are still looking for target devices whose implementation of raw MIDI complies with the MIDI standard.

Bluetooth control

The Nokia 6230 is also Bluetooth-enabled. This allowed us to modify the volume and pitch of a moving sound source by remote control. It will also allow multiple parts played on different handsets to be synchronised or played via manual operations performed on a remote handset.

All Bluetooth communication is defined by the protocol specification and the operation of the Bluetooth stack [Java Community Process 2003]. Within the Bluetooth stack, the lowest-level communication layer accessible to a j2me application is the Logical Link Control and Adaptation Protocol, or L2CAP. Other protocols that handle data in a variety of packet formats operate via L2CAP.

While existing j2me APIs such as RFCOMM and TCS-binary allow synchronous transmission of large data packets, there are noticeable latencies when these are used to transmit small message packets such as MIDI messages.

On top of the L2CAP layer, we have found it necessary to create a customised API for Asynchronous Input Output – shown as AIO in Figure 1. This was necessary to accommodate smaller message packets within the Bluetooth Protocol. This API supports asynchronous byte-oriented protocols such as MIDI messages where data is sent in packets whose size is defined by the MIDI Status Byte.

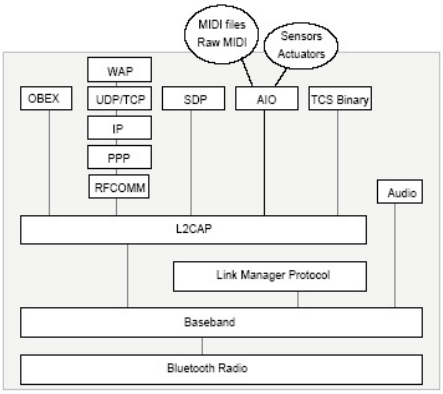

Figure 1 shows the Bluetooth stack and its relationship with external asynchronous communications devices such as sensors and actuators and MIDI.

Asynchronous Input and Output data sent or received via Bluetooth is transmitted as MIDI message packets. Data transmitted to actuators or received from sensors will also be transmitted as MIDI message packets.

A simple control MIDlet shown in Figure 2 demonstrates how asynchronous Bluetooth communication works. A momentary single-byte phone key operation performed on a master device is instantaneously transmitted to a slave device.

In this API a master can also address each slave individually before each MIDI message. A pre-byte transmitted before each MIDI message packet effectively extends the number of available MIDI channels in a single Bluetooth channel.

Figure 2 shows a single key ("8") pressed on the master device (larger phone). The corresponding key appears in the MIDlet which displays the pressed key on both the master and the slave devices.

The Bluetooth pre-byte is an 8-bit positive number that is used to address a slave device individually. It contains a code that selects a Bluetooth channel and is sent via Bluetooth immediately prior to sending the MIDI message. A positive pre-byte sent as part of Bluetooth transmission will not interfere with the way MIDI parses message packets because MIDI uses an 8-bit negative number to signify the start of a MIDI packet and determines the size of each packet by parsing each MIDI status byte. If an 8-bit negative number follows, this indicates that a Bluetooth transmission begins without a Bluetooth channel address, i.e. it is broadcast on all Bluetooth channels. An 8-bit positive Bluetooth pre-byte will therefore work with every variety of MIDI packet except for MIDI Running Status.

In order to make this fully compliant with MIDI, we have decided that the AIO recognises MIDI Running Status messages received from a non-Bluetooth source. It will relay these messages over the Bluetooth network using the shortest MIDI packet available i.e. all two- or three-byte packets or MIDI System Exclusive.

The total number of MIDI channels for a full complement of 7 Bluetooth channels is 112 (i.e. 7 x 16). In terms of mobile phones, this represents 112 individually addressable mobile phones each receiving on a dedicated MIDI channel unconstrained by operating at MIDI transmission speeds.

Composition Interface

Two public domain resources facilitate the use of mobile devices for microtonal composition. Pure Data, or PD, an algorithmic composition language developed by Miller Puckette [Puckette 1996], is used to compose for mobile phones. Scala, tuning analysis-editor-librarian software developed by Manual Op de Coul [Op de Coul 1992], is used to implement microtonal tuning.

Composers familiar with a graphical object-oriented interface can apply these skills in composing for mobile phones using PD. While it is possible to compose music for mobile phones by writing entirely in java code, a graphic composition environment seems more appropriate to help composers design and simulate the kind of interactive musical applications that will run in a mobile environment.

Scala is another public domain freeware resource that composers will find invaluable for creating microtonal compositions. Scala has a scales archive containing over two thousand tunings including both just intonation and various equal-division-of-the-octave, both historical and experimental. Moreover, Scala allows a user to create command scripts that export its tuning resources to external programs including PD.

However, in order to make a more accessible composition interface for mobile devices it was also necessary to develop a desktop Java application that cross-compiles from PD to j2me.

pd2j2me

pd2j2me allows musical applications composed for mobile phones to be simulated in a desktop environment [Schiemer and Havryliv 2005]. It consists of two objects. One reads a PD file and translates it into java. The other is a PD object that simulates the functions of a mobile phone handset.

Given the limitations of keypad, screen and available memory in a typical j2me device, it was decided to build a cross-compiler that converts PD files into java rather than a run-time PD interpreter for j2me.

A composer working on a desktop PC writes a PD file in which a number of mobile phones are configured. The operation of each phone can be simulated and tested before pd2j2me imports code from the

working PD patch and compiles a j2me source file which is finally downloaded into each handset.

Like PD, the pd2j2me compiler displays interconnectable graphic objects. The compiler supports:

- simple j2me expressions (e.g. var = arg + const);
- calls to pre-built generic j2me classes that support more complex objects, e.g. line, metro, delay, and;
- simplified audio, MIDI, Bluetooth wireless connection and user IO

PD tuned from Scala

Automated Scala commands allow a composer to access Scala's microtonal resources and export them to PD. The following PD patch takes tuning values directly from Scala's tuning table.

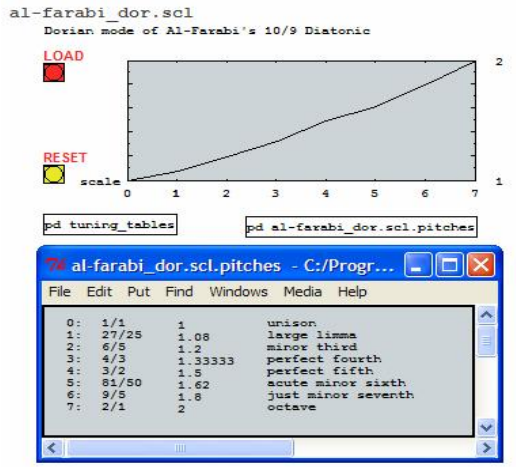

Figure 3 Tuning values exported from Scala are accessed in PD as linear factors.

Tuning values in the PD patch are shown as linear factors which are formed by dividing the numerator by the denominator of the just tuning ratio. When multiplied by a reference frequency a set of linear factors will produce the pitches of a scale. Scala will supply linear factors for both just intonation and equal tempered scales

Conclusion

Development of new applications for mobile phone technology is driven predominantly by concerns of the corporate world. We hope this technology whose origins are derived from Antheil's musical instrument design [Price 1983] will once more be driven by communities of musicians just as MIDI users did two decades ago.

The mobile phone will become less like a virtual shopping trolley and more a playground for communities of musical hunters and gatherers to explore the world's tuning systems and understand the diversity of human imagination that created them.

Acknowledgments

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

References

Schiemer, G., Sabir, K. and Havryliv, M. 2004 "The Pocket Gamelan: A J2ME Environment for Just Intonation" *Proceedings of the International Computer Music Conference*, Miami, USA

Schiemer, G., Alves, W., Taylor, S. and Havryliv, M. 2003 "The Pocket Gamelan: building the instrumentarium for an extended harmonic universe" *Proceedings of the International Computer Music Conference*, Singapore

Java Community Process 2002a Mobile Information Device Profile (MIDP) 2.0

Java Community Process 2003 Connected Limited Device Configuration (CLDC) 1.1

Java Community Process 2002b JSR-000082 Java APIs for Bluetooth http://jcp.org/aboutJava/communityprocess/final/jsr082/index.html

Puckette, M. 1996: Pure Data software V0.38 http://at.or.at/hans/pd/installers.html

Op de Coul, M. 1992 Scala available at http://www.xs4all.nl/~huygensf/scala/

Schiemer, G. and Havryliv, M. 2005 "Pocket Gamelan: a Pure Data interface for java phones" *Proceedings of the New Interfaces for Musical Expression Conference*, Vancouver, Canada

Schiemer, G. and Havryliv, M. 2005 "Pocket Gamelan: a blueprint for performance using wireless devices" accepted for *Proceedings of the International Computer Music Conference,* Barcelona, Spain

Schiemer, G. and Havryliv, M. 2004 "Wearable Firmware: The Singing Jacket" *Proceedings of the Australasian Computer Music Conference,* Wellington, New Zealand

Price, R 1983: Further Notes and Anecdotes on Spread-Spectrum Origins see IV. Shortly before Pearl Harbour: The Lamarr-Antheil Frequency-Hopping Invention, *IEEE Transactions on Communications Vol. COM-31*, No.1 pp. 89-91