1-1-2007

Pocket Gamelan: tuning microtonal applications in Pd using Scala

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

M. O. de Coul
Huygens Fokker Foundation, The Netherlands

Pocket Gamelan: tuning microtonal applications in Pd using Scala

Greg Schiemer
Sonic Arts Research Network
Faculty of Creative Arts
University of Wollongong, 2522
Australia
schiemer@uow.edu.au

Manuel Op de Coul
Huygens Fokker Foundation
Muziekgebouw aan 't IJ
Piet Heinkade 5, NL-1019 BR Amsterdam
The Netherlands
coul@computer.org

Abstract

Microtonal tuning has been a characteristic common to many musical traditions yet despite a growing awareness of these traditions among many musicians today, a single system of tuning based on the twelve-note equal division of the octave continues to dominate development of multimedia applications. This paper describes a new software tool developed to export and document microtonal scales for use in computer music and multimedia composition. The tool was developed as a command script written by the first author using an editor, librarian, and analysis tool for musical tunings known as Scala, written by the second author. The tool called scaleplayer.cmd allows tuning to be exported from Scala to Pure Data, an environment for algorithmic composition where novel purpose-built performance interfaces can be prototyped easily. The tool allows composers to interact with thousands of historical and novel scales and to develop a user interface based on a new understanding of the tuning characteristics being explored. Pure Data has already been used to create performance interfaces for the Pocket Gamelan, a project that has allowed new microtonal tunings to be implemented and performed using Blue-tooth enabled mobile phone technology.

Introduction

The Pocket Gamelan project was launched by the first author in 2003 to address the challenge of composing music for a mobile computing environment. Central to this project was the development of an interactive musical performance interface that allowed non-expert performers to perform microtonal music using mobile phones. Several mobile performance scenarios were implemented as a way to explore the musical legacy of historical tuning systems as well as the tuning systems first explored by composer and theorist Harry Partch and later extended through the work of contemporary tuning theorist Erv Wilson (Schiemer and Havryliv, 2006).

Two new microtonal works for mobile phones have been created and performed at UK Microfest, 2005, NIME06, 2006 and Microfest, 2007. As part of the composition procedure, Pure Data (Pd) files containing microtonal data were created and documented using Scala. Another purpose-built tool, developed by Mark Havryliv, was then used to translate Pd files into j2me, a format suitable for java phones (Schiemer and Havryliv, 2005). Prior to performance realisation using multiple phones, performances were emulated and auditioned using Pd files running on a single desktop machine. The second author added enhancements to Scala that allowed tuning data to be exported and documented as text files readable as Csound and Pd files. These enhancements were added in version 1.7 to allow text files produced by Scala to be read as Csound files and further enhanced in version 2.2o to be read as Pd files.

Scala

Scala is cross platform freeware designed

> "for experimentation with musical tunings, such as just intonation scales, equal and historical temperaments, microtonal and macrotonal scales, and non-Western scales" (Op de Coul 2007).

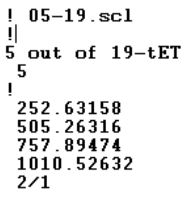
Written in the programming language Ada, Scala has a graphical user interface as well as a command line interface to over 450 functions for scale analysis and manipulation. It has an extensive knowledge base that includes a scale archive containing more than 3400 scales. It recognises more than 1100 musical modes, more than 500 chords and supports more than 400 note naming systems. It offers flexible keyboard mapping, plays scale tones via the soundcard and exports tuning data to a variety of synthesizers with an internal tuning table. It can create MIDI files from a microtonal score, retune existing MIDI files or relay real-time MIDI messages. Its functionality is extensible through the use of command scripts and screen output can be captured to text files.

Scala Command files

Scala commands are sequenced using scripts known as command files. The command file type is .cmd. Screen output is first captured by opening a file, displaying an output string on the screen then closing the file. By observing syntax used in other programs such as Csound and Pd, command files may export tuning information to other programs.

Scala Scale file format

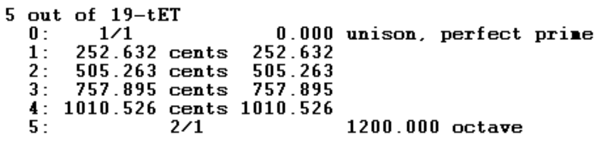
Scala uses an ASCII-based scale file format to store scales. The scale file type is .scl. This format has gradually been adopted as a de facto standard for microtonal scales. Exclamation marks precede comment lines. The first non comment line in a scale file contains a short description of the scale; the second line specifies the scale size; these are followed by a list of pitch values expressed in ratios or cents, as shown in Figure 1.



```
! 05-19.scl
!|
5 out of 19-tET
 5
!
 252.63158
 505.26316
 757.89474
 1010.52632
 2/1
```

Figure 1. *Scala scale file format viewed as text file.*

Once a scale file is loaded into scale memory, other commands are used to operate on scale data. Other scale attributes such interval size, and where applicable, historical interval names, are displayed as shown in Figure 2, using SHOW or 'F6'.



```
5 out of 19-tET
  0:     1/1             0.000 unison, perfect prime
  1:   252.632 cents   252.632
  2:   505.263 cents   505.263
  3:   757.895 cents   757.895
  4:  1010.526 cents  1010.526
  5:         2/1          1200.000 octave
```

Figure 2. Scale file viewed using 'Show'

Pd-scale-player.cmd

Pd-scale-player.cmd is a Scala command file that generates tuned Pd files. It is included as part of the library of command files with releases from Scala V2.2o or thereafter.

A user first selects a scale from the scale archive and loads it into scale memory. Running pd-scale-player.cmd will generate a Pd file using tuning data in scale memory.

The command is invoked using Scala's GUI by selecting File, Shift+Alt+@ command file name. Alternatively, in the Scala command line interface, the user may type:

```
@ pd-scale-player.cmd
```

The first line in the scl file is used to document the Pd patch. The second is used as a parameter to generate objects in a Pd patch that match the size of any scale automatically. This allows the command to generate Pd files easily using scales of any size. To create a new scale in Pd, the user selects a new scale in Scala then re-runs pd-scale-player.cmd.

Echo commands

The ECHO command in Scala is used to display data captured in Pd files. Lexical functions were added to the ECHO command at the request of the first author so that external files could be created using data exported directly from scale memory. In Lexical functions were introduced in Scala v1.7 to export pitch information into Csound files. Some of these have also been used to create Pd files.

Lexical functions of ECHO are preceded by % and include in parentheses pitch parameters that convert values in scale memory into text output. Values in parentheses can be a pitch memory if preceded by a $; or they can be a scale degree if preceded by a %; or they can be literal, in which case they are not preceded by any token.

Lexical functions – v1.7

Lexical functions in Scala version 1.7 include:

%cents(pitch)
Gives the cents value.

%factor(pitch)
Gives the linear value.

%hertz(pitch)
Gives the frequency in Hertz relative to the base frequency.

%image(pitch)
Gives the ratio of a rational pitch or cents value of a floating pitch.

%midi(pitch)
Gives the fractional MIDI note number relative to the base frequency.

%name(pitch)
Gives the interval name of a rational pitch.

%octcps(pitch)
Gives the Csound/SAOL oct value relative to the base frequency.

%primes(pitch)
Gives the prime factorisation of a rational pitch.

Lexical functions – v2.2o

Additional lexical functions were introduced in version 2.2o to create Pd files:

%den(pitch)
Gives the denominator of a rational pitch.

%desc(scalenr.)
Gives description belonging to the scale memory.

%listfactor(scalenr.)
Gives the scale as a list of linear factors starting at degree 1. If necessary, start with 1.0 for degree 0.

%n(scalenr.)
Gives the number of notes in a scale memory. Adding and subtracting is also possible, like %n-1(0).

%num(pitch)
Gives the numerator of a rational pitch.

%scl(appendix)
Gives the name of the last scale file loaded, regardless to which scale memory. The appendix may be the name of a scale file or an empty string.

Generating Pd source code in Scala

When a Pd file is read using a standard text editor, Pd objects appear as a line beginning with hash (#). A Scala command file may be used to generate a Pd file if every # is preceded by ECHO followed by a space. This can be achieved using the search and replace function in any standard text editor.

Creating a Pd parent canvas

The first canvas, shows a Pd scale template SCALE.PD created using Scala's FILE and CLOSE command. All ECHO commands between FILE and CLOSE result in output that appears on the Pd parent canvas. The lexical function %desc is used to describe the scale. Additional Pd patches are placed on the parent canvas between the second ECHO and CLOSE.

```
FILE SCALE.PD
!
! Main Canvas
!
! Below this point observe
! PD syntax when using ECHO
!
ECHO #N canvas 24 5 750 350 10;
ECHO #X text 5 16 %desc(0);
!
! insert other Pd objects here
!
CLOSE
```

Example 1

In this example, the first ECHO #N creates a new canvas, in this case the parent canvas. The first four variables define the area. The 1st and 2nd are the vertical (24) and horizontal (5) coordinates at the top left; the 3rd and 4th are the vertical (750) and horizontal (350) coordinates at the bottom right. The 5th variable is the default font size (10) for the parent canvas and all embedded canvases.

The second ECHO command writes a comment in Pd describing the scale last loaded into scale memory 0. This is displayed at a position defined by horizontal (5) and vertical (16) coordinates. The scale description is supplied by the scale archive.

Pd – tuning data

As the primary focus of the project has been just intonation, tuning has been implemented using linear factors formed when numerators are divided by denominators. Each note in the scale is tuned by multiplying its linear factor by the base frequency. Though linear factors are normally associated with just intonation, Scala allows linear factors to be used to express non-just musical intervals.

Figure 3 shows the Pd sub-patch embedded in pd-scale-player.cmd which uses linear factors to tune oscillators. The oscillators are played by the note keyboard that is played by an algorithmically generated sequence. The interface used to play the file allows sequences to be transposed by octaves or using modes of the microtonal scale. The interface is shown in Figure 4.

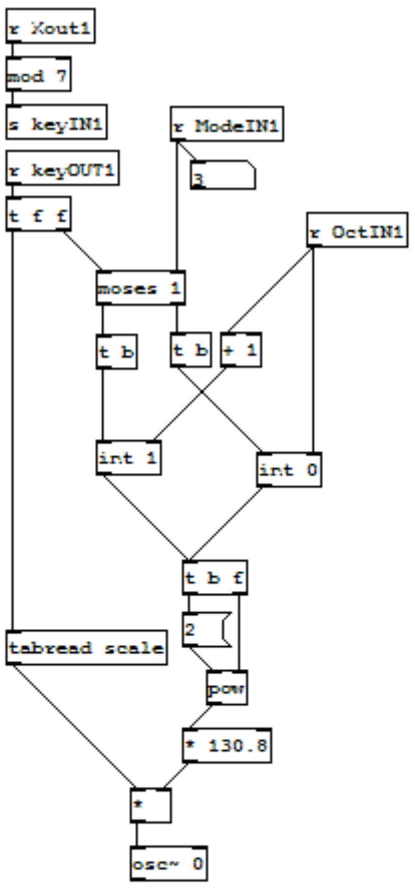


Figure 3. Oscillator in Pd tuned using linear factors

Example 2 shows the embedded object used to display the contents of a tuning array. The array is loaded with a scale and named using the lexical function %scl(). The first ECHO command forms the canvas of the array object at a point specified by horizontal (45) and vertical (68) coordinates. The canvas is then overlaid by the graph object in the last ECHO command.

```
! Canvas 2 - display scale
!
ECHO #X obj 45 68 cnv 15 254 99 empty
empty %scl() -40 -60 0 16 -212343 -
258699 0;
!
! Create array of linear factors
!
ECHO #N canvas 0 0 450 168 graph1 0;
ECHO #X array scale %n+1(0) float 1;
ECHO #A 0 1 %listfactor(0) \:;
ECHO #X coords 0 2 %n(0) 1 256 100 1;
!
ECHO #X restore 44 67 graph;
```

Example 2

When it is open, the canvas has a default hot spot size of 15, is 254 units wide, 99 units high. It has three symbolic properties, two of which are undefined (i.e. empty). The 3rd is the canvas label containing the name of the scale last loaded into scale memory from the scale archive. It is defined using the lexical function %scl().

The next two variables define the coordinates of the canvas label relative to the canvas, (-40 horizontal, -60 vertical); the 3rd variable selects the font (0 = Courier New); the 4th variable selects font size (16); the 5th and 6th variables define background and font colour (-212343 = grey, -258699 = red). The final variable 0 is unused on the parent canvas.

The second and third ECHO command creates the array for storing and displaying tuning data. The 2nd ECHO defines the visible geometry of the array. While the canvas is open, the 1st and 2nd variables are the vertical (0) and horizontal (0) coordinates at the top left; the 3rd and 4th are the vertical (450) and horizontal (168) coordinates at the bottom right. The 3rd ECHO creates an array of floating point numbers called 'scale'. Array size is defined by the lexical function %n+1(0) and refers to the size, plus one, of scale memory 0. By adding one, both the unison and octave are each aligned with opposite boundaries of the graph.

The 4th ECHO loads tuning values from scale memory 0 into the array using the lexical function %listfactor(0) which presents tuning data as a list of linear factors. The first factor is stored at location 0, the first location in the array. Because unison is not included in the list of linear factors created by Scala, 1, the value for the unison must be inserted before the list of factors. The list is terminated with backslash (\) followed by colon (:) to comply with message syntax used in Pd.

The 5th ECHO defines the range of values used to display tuning data within the geometry of the scale canvas. The first two variables are values at the horizontal (0) and vertical (2) coordinates in the upper left corner of the display; the next two are the values at the horizontal (%n(0)) and vertical (1) coordinates in the lower right corner of the display. The lexical function %n(0) divides the horizontal domain into a number of discrete points equal to the size of the scale in scale memory 0. The vertical domain covers the range of linear factors between the unison (1) and the octave (2) because all just ratios, by convention, are expressed within the range of one octave. The remaining three variables affect the size of the graph object. The 5th variable (256) determines the width of the graph, the 6th variable (100) determines its height and the 7th variable (1) determines whether the graph object is open or closed.

Pd – tuning documentation

Exporting tuning documentation from Scala to Pd presented a special challenge. This was largely due to the way Pd processes strings and represents numbers in floating point. It is a simple matter for Scala to create a text file that can be used to display scale specifications in other programs. It is not a trivial matter to display this clearly in Pd as text is interpreted in a way that does not always allow Pd to display a text file literally.

In Pd some words are reserved for use as part of a token. For example, a comment begins with the token '#X text'. This is followed by two variables describing the vertical and horizontal coordinates of the character string displayed. Finally, this is followed by the string itself.

Example 2 displays the specifications of any scale exported from Scala to Pd. These include not only the name of the scale, displayed using %scl(), but data about each pitch such as intervals formed relative to the unison, and the historical names of intervals where these exist. Intervals formed are expressed as tuning ratios or cents and displays using the lexical function %image (pitch) which automatically expresses just intervals as fractions and non-just intervals as cents. Scala also automatically identifies historical interval names and these are displayed using %name(pitch). In addition to these, new lexical functions were created to display numerators (%num()) and denominators (%den()).

The problem of documenting scale information in Pd has been addressed by using Scala scale memory as general purpose memory. This allowed us to create and store sets of vertical and horizontal coordinates for creating a coherent display of Pd comments. Working with Scala scale memory in this way was somewhat similar to working with general purpose registers when programming in assembler. Scala uses commands like COPY, MOVE, CLEAR, ADD, etc. which operate on scale memories. Unlike a general purpose register which is a single unit of stored data, a scale memory is a complex array of data on which many operations may be performed iteratively whenever a Scala command is performed.

Tuning documentation is displayed as rows of pitches arranged in four columns. Iteration is used to display the notes of the scale in successive rows. The HARMONIC command was used to produce a number sequence. Normally this command is used to create harmonic scales, but was used here as a way to number the lines in a text display of tuning information. The program code in example 3 is an embedded command file called 'ordinal.cmd'. When this is run later (in example 4), the HARMONIC command will produce a number sequence between 1 and the number of notes in any scale loaded into scale memory 8. The sequence created is stored in scale memory 0. The size of the scale in scale memory 8 represents the number of lines in the display.

```
! Generate PD Scale template
!
FILE ordinal.cmd
ECHO ! ordinal.cmd
ECHO ! This command is a template gen-
erated by PD-Scale-Player.cmd
ECHO !
ECHO HARMONIC 1 %n(8)
CLOSE
```

Example 3

Four display columns are aligned using horizontal coordinates that are entered using the INPUT/LINE command shown in example 4. The four variables 40 80 200 and 305 in the following line, are the coordinates of the four columns; these are stored by default in scale memory 0, and then saved in scale memory 5.

```
! Display coordinates for 4 columns
!
INPUT/LINE
40 80 200 305
COPY 0 5
CLEAR 0
```

Example 4

Example 5 runs 'ordinal.cmd', the command file generated in example 5. INSERT is used to increase scale size by 1. The extra number allows the unison to be listed as ordinal 0. This is a 'housekeeping' measure to display the first note correctly. Ordinals are then saved in scale memory 1.

It is then necessary to space each row so that characters are displayed in 10 point. To do this, the first line must start at 0, followed by the 2nd at 11, the 3rd at 21, the 4th at 33, and so forth. The ITERATE command is used to achieve this. The command modifies ordinals to create the sequence 0, 11, 22, 33, etc. This produces lines spaced 11 points apart. Y-coordinates are then copied into four separate scale memories, one for each column. This is necessary because the interative process used to generate each column is destructive.

```
! Generate ordinals
!
@ordinal.cmd
INSERT 1 1
COPY 0 7
SHOW/LINE 0
ECHO
!
COPY 0 1
CLEAR 0
!
! Define height of each row
! vertical spacing - 11 points
!
ITERATE/scale "INSERT 1 11" 1
ADD 1
ECHO Y-coordinates
SHOW/LINE 0
!
! Store column co-ordinates in
! scale memories 1 2 3 and 4
!
COPY 0 1
COPY 0 2
COPY 0 3
COPY 0 4
```

Example 5

Finally, ITERATE is used to display the salient features of any scale as comments in Pd using coordinates generated in Examples 4 and 5. Comments are aligned in four columns. In the 1st, note order is displayed; in the 2nd, the image (just ratio or cents); in the 3rd, linear factors; and in the 4th, historical interval names, where these exist.

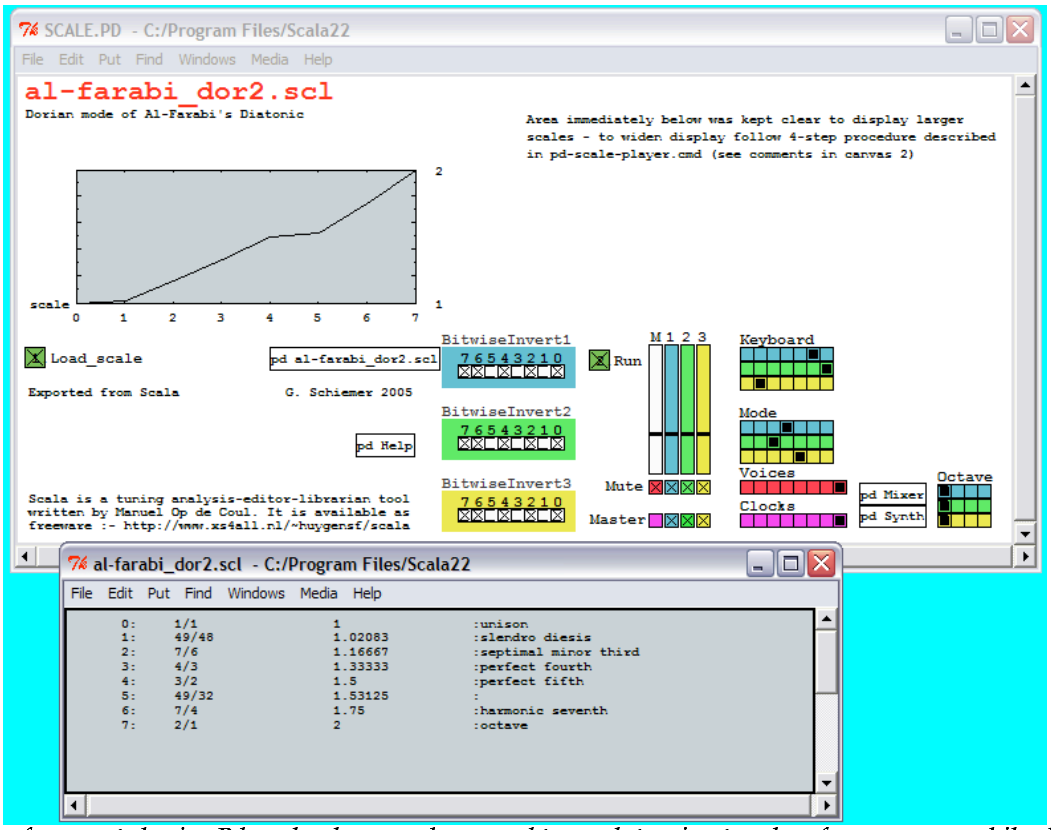
All rows except the 1st are generated iteratively. This was necessary in order to represent the first scale degree as ordinal 0. All numbers in scale memories are represented as fractions; coordinates are stored as a numerator over the denominator 1. This makes it necessary to read only the numerator and write it for Pd to display. The lexical function %num(pitch) was used for this purpose. All lexical functions shown in Example 6, specify pitch using scale-degree followed by scale-memory.

Conclusion

Exporting tuning data from Scala to Pd became complicated by their different idiosyncrasies. Yet despite these complications, well documented Pd patches serve to identify the landmarks composers need for microtonal exploration. Command files in Scala also give control over the layout in Pd as text editing makes it possible to align Pd objects precisely. Moreover, Scala command files provided an extremely stable development environment for prototyping microtonal mobile phone applications using Pd. The techniques described have been applied for Csound source files as well as Pd files. It is possible also to generate microtonal source files for other computer music languages such as MaxMSP and Supercollider, or using Scala to create microtonally tuned musical applications in java.

Acknowledgements

The Pocket Gamelan project was funded by an ARC Discovery Project 2003 – 2005.



al-farabi_dor2.scl

Dorian mode of Al-Farabi's Diatonic

Area immediately below was kept clear to display larger scales - to widen display follow 4-step procedure described in pd-scale-player.cmd (see comments in canvas 2)

Exported from Scala

G. Schiemer 2005

Scala is a tuning analysis-editor-librarian tool written by Manuel Op de Coul. It is available as freeware :- http://www.xs4all.nl/~huygensf/scala

```
0:    1/1       1          :unison
1:    49/48     1.02083    :slendro diesis
2:    7/6       1.16667    :septimal minor third
3:    4/3       1.33333    :perfect fourth
4:    3/2       1.5        :perfect fifth
5:    49/32     1.53125    :
6:    7/4       1.75       :harmonic seventh
7:    2/1       2          :octave
```

Figure 4. Interface created using Pd-scale-player.cmdwas used to emulate microtonal performance on mobile phone

```
! Display scale specifications
!
! First line of each column is read out separately
! All other lines of each are read out iteratively
!
! column 1 - Ordinals
!            x-coord 1  y-coord 1
ECHO #X text %num(%1%5) %num(%i%1) 0:;
ITERATE/SCALE "echo #X text %num(%1%5) %num(%i%1) %num(%i%7):;" 8
!
! column 2 - JI ratio (or cents)
!            x-coord 2  y-coord 1
ECHO #X text %num(%2%5) %num(%i%2) %image(%0%0);
ITERATE/SCALE "echo #X text %num(%2%5) %num(%i%2) %image(%i%0);" 8
!
! column 3 - Linear factors
!            x-coord 3  y-coord 1
ECHO #X text %num(%3%5) %num(%i%3) %factor(%0%0);
ITERATE/SCALE "echo #X text %num(%3%5) %num(%i%3) %factor(%i%0);" 8
!
! column 4 - Historical names
!            x-coord 4  y-coord 1
ECHO #X text %num(%4%5) %num(%i%4) :%name(%0%0);
ITERATE/SCALE "echo #X text %num(%4%5) %num(%i%4) :%name(%i%0);" 8
ECHO #X restore 190 200 pd %scl();

Example 6
```

References

Op de Coul, M. 2007 Scala Home Page http://www.xs4all.nl/~huygensf/scala/

Schiemer, G., and Havryliv, M. 2006 "Pocket Gamelan: tuneable trajectories for flying sources in *Mandala 3* and *Mandala 4*" Proceedings of the 2006 International Conference on New Interfaces for Musical Expression (NIME06), Paris

Schiemer, G., and Havryliv, M. 2005 "Pocket Gamelan: a Pure Data interface for mobile phones" Proceedings of the 2005 International Conference on New Interfaces for Musical Expression (NIME05), Vancouver, BC, Canada.