U.B.O
UFR Sciences et Techniques

LÉON Fabien

Département Informatique
IUP Ingenierie Informatique
Licence 3, année 2008/2009

Brain Computer Music Interface using Harmony Space

ICCMR - University of Plymouth
31th March - 19th June
Supervisor: Eduardo R. Miranda

Brain Computer Music Interface using Harmony Space

Acknowledgment

I would like to acknowledge Eduardo R. Miranda who had proposed this subject, Phillippe Le Parc to make me in contact with him. François Monin for monitoring and interest in my project. Obviously also the PhD student working in the laboratory and Simon Holland for their cordial welcome.

I would like to thank the IUP informatique and the Plymouth University for make this kind of work placement possible.

Summary

Table of Contents

INTRODUCTION

I am in the last year of a three year Degree in computer science. I do it in France in the IUP “Ingerierie Informatique” of UBO. During this year the have to do a three months work placement. I wanted to do it abroad, and I choose the ICCMR lab of the Plymouth University.

ICCMR means Interdisciplinary Centre for Computer Music Research. That's a laboratory managed by Eduardo R. Miranda. Poeple who work there do research at the cross road of music and computer science.

The project that I work on, is at the cross roads of neurobiology, engineering sciences and music. It consist in music composition with brain signal. Research has been done on this subject, in the lab and over the world. But our solution should be cheaper. That's a good point to be used in musical therapy, for the people with physical disabilities. It can be also used for art performance.

I. THE PROJECT

Composition of music by brain need two essential things. One for extracting information from the brain, and one to compose music. For the first one, other research carried out in the lab, have led to use a device from MindPeak. This choice is due to the fact that use a non-invasive method and because it is cheaper than other device use the same method. With regard to the music composer, we will use Harmony Space, a software developed for build music chords according to rules from musical theory.

My role in the project consist to find a way to make the both part work together. To do that I needed to understand how they work. So I was studying about the MindPeak software, and I was go to Milton Keynes to meet Simon Holland, the developer of the harmonic space.

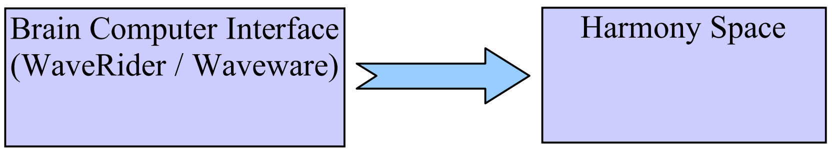


Diagram 1. The ensemble

1. WaveRider and WaveWare

The ensemble for brain information extracting from MindPeak is composed of WaveRider and WaveWare, respectively the hardware and software.

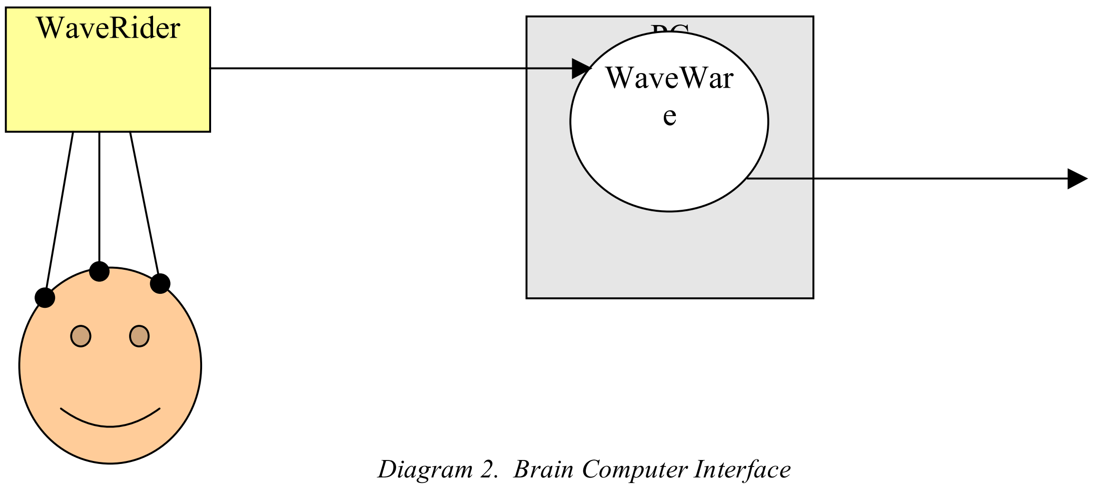


Diagram 2. Brain Computer Interface

Waverider and EEG

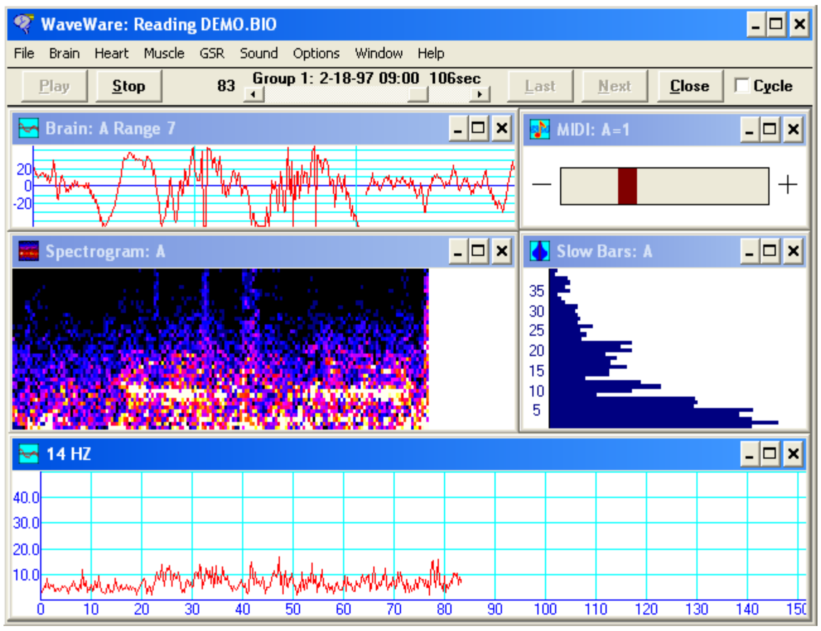
The box has 4 channels and a GSR specialize input. Wareride can measure the low electricity activity of brain, muscles and with the GSR, the resistance of the skin. The biological data thus collected are then sent to the computer through the serial port.

Picture 1. WaveRider

The method to measure the brain activity is called Electroencephalography. That's the only one we have to use. EEG is measured as a voltage difference between two or more electrodes on the surface of the scalp. It's a difficult signal to handle because it's filtering by meninges, skull and scalp. Furthermore the signals are sums of signals arising from many possible sources, including artefacts like the heartbeat and eyes blink. Contact of electrodes with the skin must be the better as possible, to get a signal that the software can scrutinize it.

Waveware

Data coming through serial port have been digitized, but they are still raw data. WaveWare will process their transformation into usable data, which can be displayed or used to generate midi sounds.


WaveWare: Reading DEMO.BIO
File Brain Heart Muscle GSR Sound Options Window Help
Play
Stop
83
Group 1: 2-18-97 09:00 106sec
Last
Next
Close
Cycle
Brain: A Range 7
MIDI: A=1
Spectrogram: A
Slow Bars: A
14 HZ

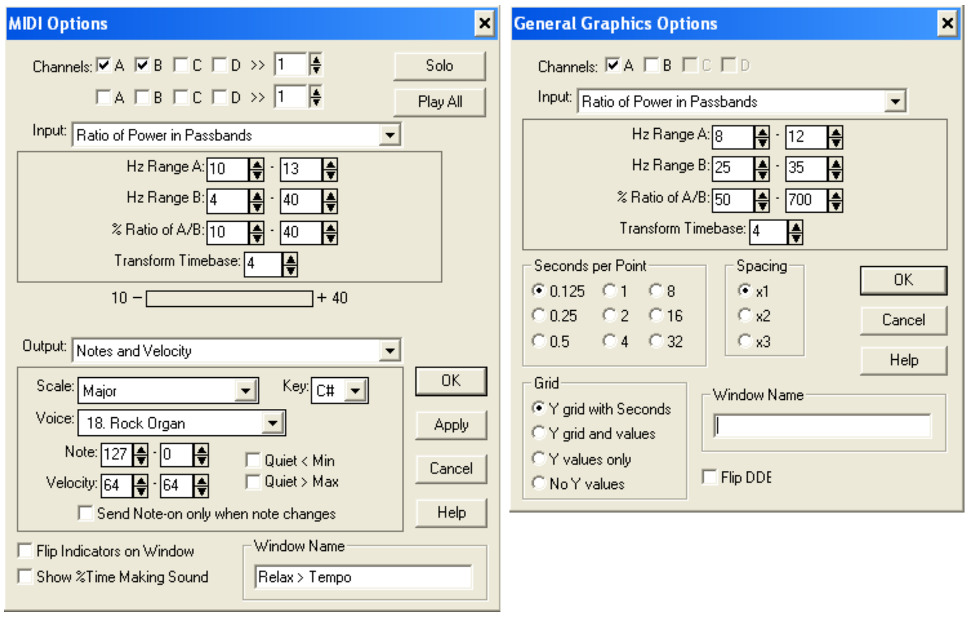
Picture 2. Screenshot of WaveWare

This screenshot show Waveware wirking with 5 boxes, which the position and setting are saved on a configuration file. There are two kinds of box, here "MIDI: A=1" is to generate sound and the others are to display.

Four kinds of display:

- Bars Graph : power on frequencies of the signal
- Spectrogram : power on frequencies according to time
- General Graph : temporal representation after processed by the following methods
  - Digital Filter
  - Ratio of two Filters
  - Reward/Suppress
  - Reward/Suppress Ratio
  - Power in passbands
  - Ration of Power in passbands
  - Average Frequency
  - Coherence (Amplitude Method)

Following picture show the box setup windows. The first one is for MIDI and have two significant section. Input: to choose what you want to transform in MIDI. Output to select which aspect of the MIDI sound will be modified by input.



MIDI Options

Channels: ☑ A ☑ B ☐ C ☐ D >> 1

☐ A ☐ B ☐ C ☐ D >> 1

Input: Ratio of Power in Passbands

Hz Range A: 10 - 13

Hz Range B: 4 - 40

% Ratio of A/B: 10 - 40

Transform Timebase: 4

10 – + 40

Output: Notes and Velocity

Scale: Major Key: C#

Voice: 18. Rock Organ

Note: 127 - 0 ☐ Quiet < Min

Velocity: 64 - 64 ☐ Quiet > Max

☐ Send Note-on only when note changes

☐ Flip Indicators on Window

☐ Show %Time Making Sound

Window Name: Relax > Tempo

General Graphics Options

Channels: ☑ A ☐ B ☐ C ☐ D

Input: Ratio of Power in Passbands

Hz Range A: 8 - 12

Hz Range B: 25 - 35

% Ratio of A/B: 50 - 700

Transform Timebase: 4

Seconds per Point: [✓ 0.125] 1 8 0.25 2 16 0.5 4 32

Spacing: [✓ x1] x2 x3

Grid: [✓ Y grid with Seconds] Y grid and values Y values only No Y values

Window Name:

☐ Flip DDE

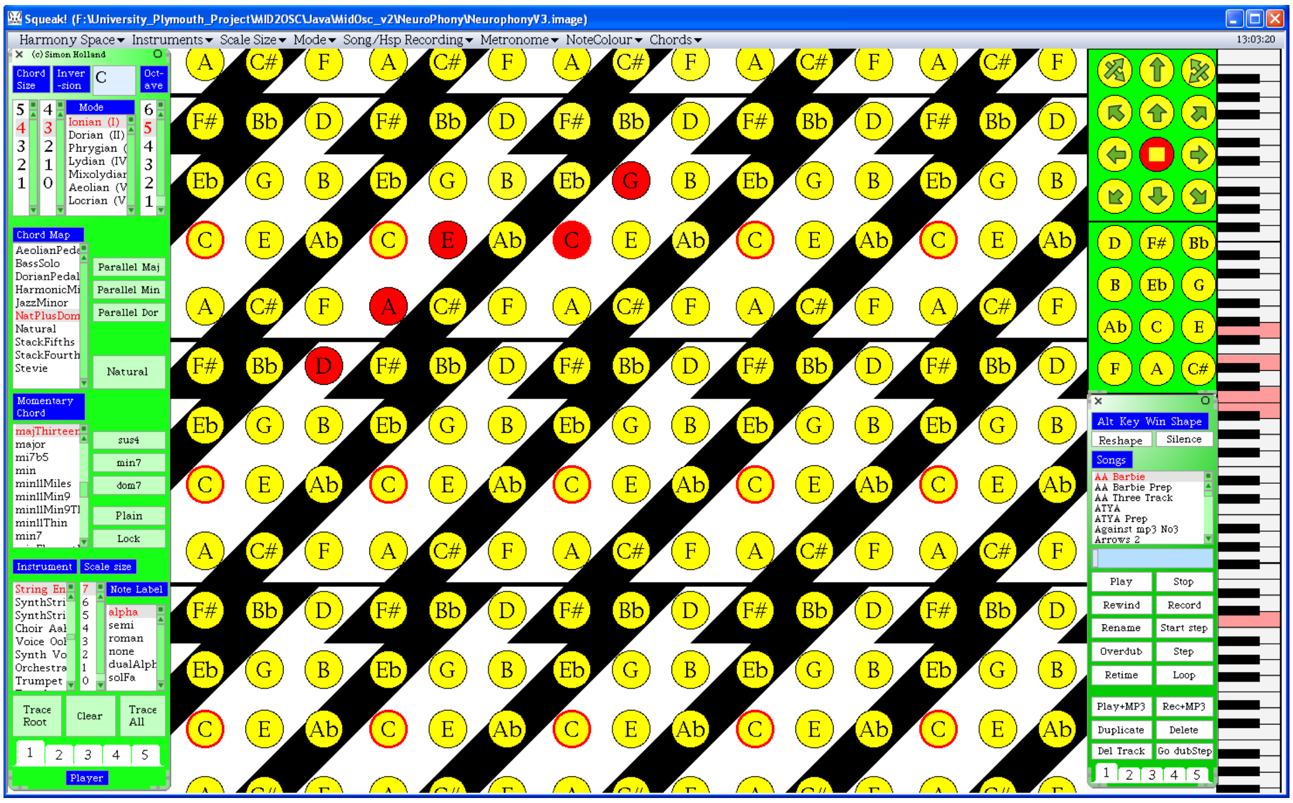
Picture 3. Waveware box setup windows

The society which developed WaveRider and WaveWare, describe it as an experimental musical instrument intended for musical performance and composition, education, and experimental computer applications. In practice it can make midi sound with information from the brain, but that doesn't sound like music. That's why we want to use Harmony Space.

2. Harnony Space

This software is developed by Simon Holland from the Open University of Milton Keynes. The aim is not to generate music with brain signal but for playing, analysing and learning about harmony.

It is a software using the research on music theory to build chord of music. It allow user to compose chords of music according to a chosen theory rules.



Picture 4. screenshot of Harmony Space

The design of the interface draws on Balzano's and Longuest-Huggins' theories of tonal harmony.

In the center there is the space that you can click on to generate chord. The left column is to choose the rules of composition, the right to record, play and move in the space, and the piano is to visualize the note used in the chords.

On the picture 4, you can see an example. In the space the are some red note which is the path following by the previous notes. On the piano, the red keys are those pressed to build the chord according to the latest pressed note.

II. RESEARCH

The fact that Squeak can't read on MIDI port, require us to find another way to make the both soft communicate. The easiest is to use an existing soft, it's that I explain in the second part. According to the result, Simon and me have found a communication protocol, it's that I explain in the last part. But for a better understanding, in a first time I will define what is MIDI and OSC.

1. What is MIDI and OSC ?

Understanding these both communications protocols has been an important part of the project like the comprehension of EEG signals. But this part doesn't contain a full description, it's just for understand the development.

Musical Instrument Digital Interface

MIDI is an industry standard protocol that enables electronic musical instruments to communicate, control, and synchronize witch each other. It does not transmit an audio signal; it transmits "event messages" such as the pitch and intensity of musical notes to play. Usually The MIDI messages are sent by a physical medium like cable. But they can pass from a software to another across a virtual MIDI port.

This protocol allows controlling multiple devices, so each message contains an information called channel. This enables the devices to know if the message affects them. There is multiple kind of MIDI message, but we will concentrate in the most useful.

Contained messages:

Note Off, Note On	=>	channel's number
		key's number (note's number)
		velocity
Control Change	=>	channel's number
		control's number
		control's value

Example of message: **1000 nnnn 0kkkkkkk 0vvvvvvv**

1000 : message identifier
nnnn : channel number (from 0 to 15)
kkkkkkk : the key (note) number (form 0 to 127)
vvvvvvv : the velocity (form 0 to 127)

Open Sound Control

MIDI has some limitation that's why some people decided to develop OSC. It works on network system, with TCP or UDP protocols. So, it can be used with the classical Ethernet device.

The *OSC Address* of an OSC Method is a symbolic name giving the full path to the OSC Method in the OSC Address Space, starting from the root of the tree.

OSC Packets: An OSC packet consists of its *contents*, a contiguous block of binary data, and its *size*, the number of 8-bit bytes that comprise the contents. The size of an OSC packet is always a multiple of 4. The contents of an OSC packet must be either an *OSC Message* or an *OSC Bundle*. The first byte of the packet's contents unambiguously distinguishes between these two alternatives.

OSC Messages: An OSC message consists of an *OSC Address Pattern* followed by an *OSC Type Tag String* followed by zero or more *OSC Arguments*.

OSC Address Patterns: An OSC Address Pattern is an OSC-string beginning with the character '/' (forward slash).

OSC Type Tag String: An OSC Type Tag String is an OSC-string beginning with the character ',' (comma) followed by a sequence of characters corresponding exactly to the sequence of OSC Arguments in the given message. Each character after the comma is called an *OSC Type Tag* and represents the type of the corresponding OSC Argument.

OSC Arguments: A sequence of OSC Arguments is represented by a contiguous sequence of the binary representations of each argument.

OSC message example: **/un/test 1000 -1 hello 1.234 5.678**

Path												Type Tag String								Argument “i”			
/	u	n	/	t	e	s	t					,	i	i	s	f	f			1000			
2F	75	6E	2F	74	65	73	74	00	00	00	00	2C	69	69	73	66	66	00	00	00	00	03	E8

Argument “i”				Argument “s”								Argument “f”				Argument “f”			
-1				h	e	l	l	o				1.234				5.678			
FF	FF	FF	FF	68	65	6C	6C	6F	00	00	00	3F	9D	F3	B6	40	B5	B2	2D

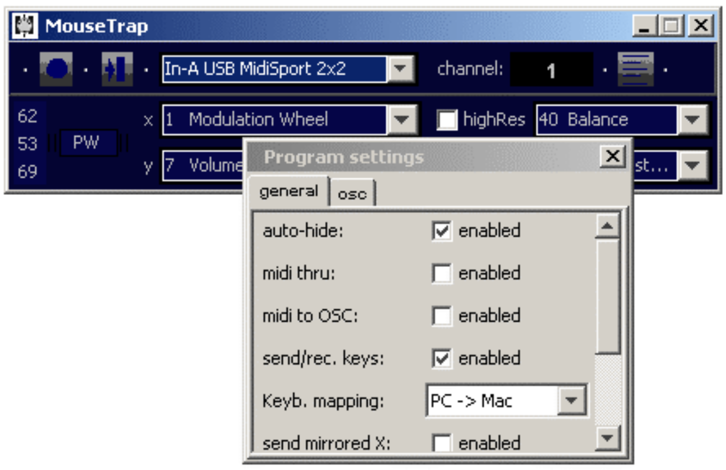
Table 1. Composition of an OSC message

2. MIDI to OSC translator

There are two possibilities for the translation. One is to build a program, and the easiest one is to find an existing software to do it. I will explore this opportunity in a first time.

MouseTrap software

After a long research on internet I was found only one software that works on Windows. The aim of this software is remote control of a computer mouse by using MIDI or OSC. But it can also translate MIDI to OSC.



In-A USB MidiSport 2x2 channel: 1

62
53 PW
69

x 1 Modulation Wheel ☐ highRes 40 Balance

y 7 Volume st...

auto-hide: ☑ enabled

midi thru: ☐ enabled

midi to OSC: ☐ enabled

send/rec. keys: ☑ enabled

Keyb. mapping: PC -> Mac

send mirrored X: ☐ enabled

Picture 5. MouseTrap software

To do it, we have to run twice the program. Set one to receive MIDI for translation, and the other to send OSC. But after various trying, it appear have a matter. Each attempt I lost control of the mouse.

This is not a good solution, so I will try to write one by myself.

Programming

To develop the software I need to choose a programming language. At first, I thought about Java because it's easy to build a graphical interface, there is a MIDI package, and I was found an OSC library (). But I was found another programming language, named PureDate. It was especially developed for music, it is a flux-based programming, and it is really easy to lean.

3. Communication protocol

When I have gone to the Milton Keynes, Simon showed me to how HSP works, and the fact that it wasn't able to compose music at the moment. And I explained to Simon how WaveWare work and which kind of information we can extract from the EEG. So, together we have to decide what is usable to adjust the composer and how we can compose.

What we can send ?

Due to the fact that I need to write a programme for the translation. We decide to use it to pre-process the information from Waveware. We also decide to start with a simple one, just for improve the chain of software. That means we will use simple composition rules, and extract just some information from the EEG.

After a discussion about the meaning of some information from the EEG, we decide to control the composer with just three informations that is the volume, the tempo and the nicety.

Between WaveWare and MIDI/OSC

For these three variables I have to find an aspect of the EEG that can be more or less controllable by the user.

I have found what I can use in the documentation of Waveware:

Volume → Alpha/Beta

Tempo → relax: ratio of power in passbands, between 10-13Hz and 4-40Hz [manual p.76]

Nicety → concentration: reward/suppress Ratio, between beta and theta wave [manual p.100]

The first information according to the tempo, changes the note's number of MIDI channel 1 between 0 and 127 to made a period between 250 and 1270ms

The second information according to the volume, changes the note's number of MIDI channel 2 between 0 and 127.

The third information according to the nicety, change the note's number of MIDI channel 3. It is interpreted like that: notes less than 60 means nice and notes upper than 60 means nasty.

between MIDI/OSC and HSP

The MIDI notes will be interpreted by the translator and will be sent in three OSC messages like that:

"/hsp/nice" path send with a boolean argument to say to HSP what kind of music playing.

"/hsp/event" path send to HSP like a tempo. (one event equal one note played)

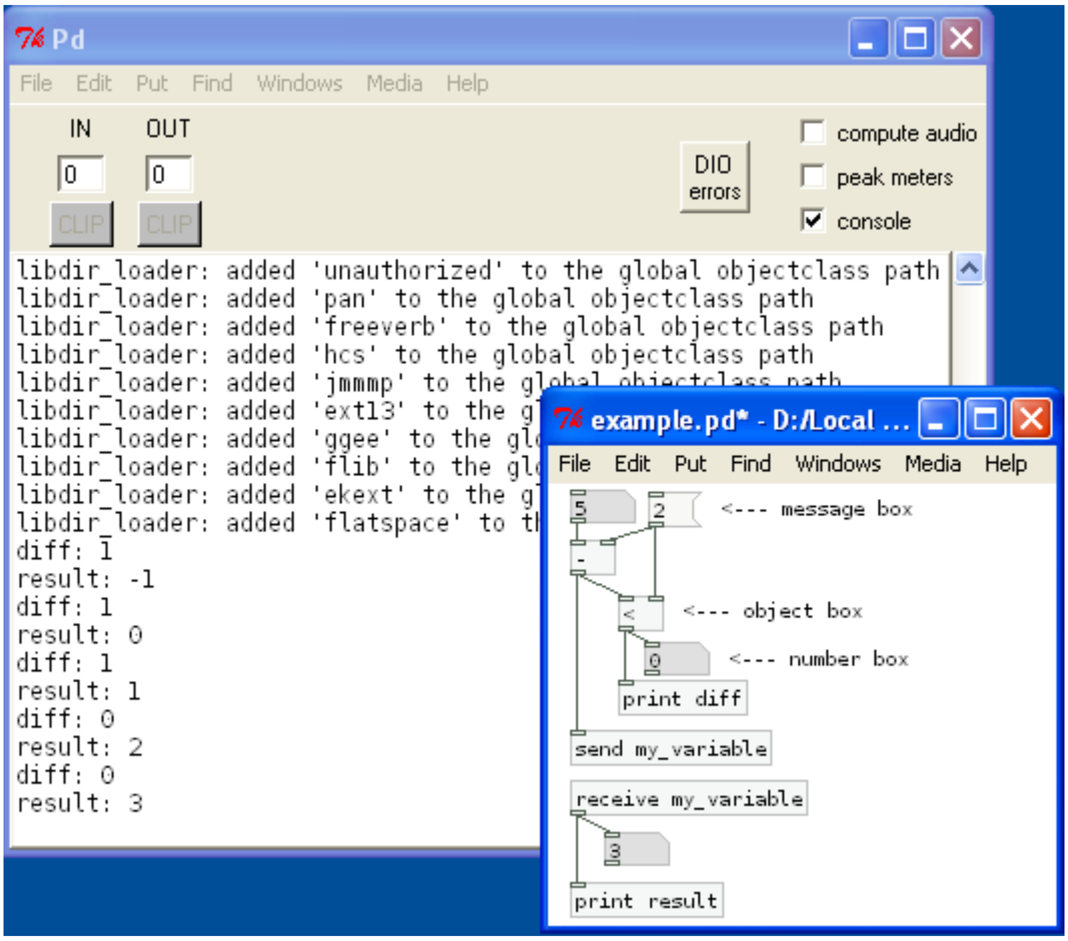
"/hsp/volume" path send with an interger argument for control the volume.

III. REALIZATION

According to the research I have learnt the graphical programming language Puredata. So, after explain how PureData and my program work, I'll explain why finally I have written a java program.

1. PureData Version

PureData is defined like a real-time graphical dataflow programming environment for audio, video, and graphical processing.



```
libdir_loader: added 'unauthorized' to the global objectclass path
libdir_loader: added 'pan' to the global objectclass path
libdir_loader: added 'freeverb' to the global objectclass path
libdir_loader: added 'hcs' to the global objectclass path
libdir_loader: added 'jmmmp' to the gl
libdir_loader: added 'ext13' to the g
libdir_loader: added 'ggee' to the gl
libdir_loader: added 'flib' to the gl
libdir_loader: added 'ekext' to the g
libdir_loader: added 'flatspace' to th
diff: 1
result: -1
diff: 1
result: 0
diff: 1
result: 1
diff: 0
result: 2
diff: 0
result: 3
```

```
print diff
send my_variable
receive my_variable
print result
```

Picture 6. Example of Pd program

The programming environment is composed of a main Pd window, which contain the logs, and possibly one or more programming window. Programming consists to put boxes and link them. And due to the fact that Pd is a real time programming, you have just to switch between “edit” and “run” mode to run it.

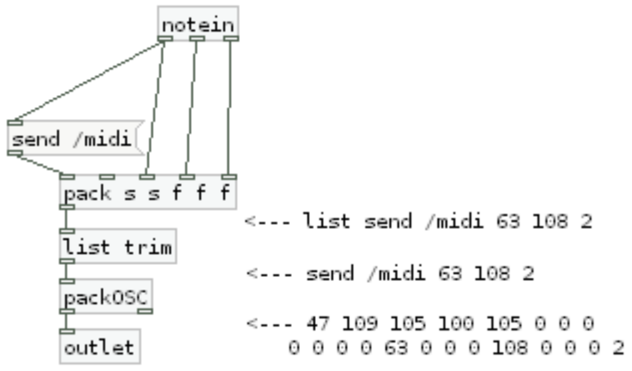
There are four types of boxes:
•In picture 6 you can see three number boxes, that's GUI box, because in “run” mode you can affect it by click on. In the GUI group there are also the toggles buttons, the sliders and other that we will see after.
•The message boxes interpret the text as a message to send whenever the box is activated, that can be by an incoming message or with the mouse.
•All the others item on picture 6 are objects boxes. This is the most useful one, the behaviours and the number of input/output of them, is defined by what you typing on. Can type like in my example “-”, “<”, “print”, “send” and “receive”. After some of then I have writing other thing, it's like argument. We can also write “* 4”, to build an object for multiply by 4.
•The last kind of box, doesn't look like box, it's the comment.

Now we have the basics knowledge to understand PureData programming. So, I will explain the functioning of my program and also the objects used.

MIDI to OSC translator

Before according with Simon about the protocol, I had written a very simple program to test if Pd is so easy that I read and if the translation is possible. For each MIDI note received, it build an OSC message which the path is '*/midi*' and the arguments are MIDI note number, velocity and channel.

Example: MIDI → *note:63, velocity:108, channel:2*
OSC message → '*/midi 63 108 2*'



```
<--- list send /midi 63 108 2

<--- send /midi 63 108 2

<--- 47 109 105 100 105 0 0 0
    0 0 0 0 63 0 0 0 108 0 0 0 2
```

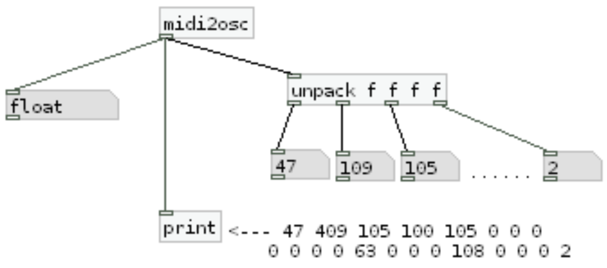
Picture 7. Simple Pd MIDI to OSC translator

To understand how it work I have commented the code with what is sent by the boxes according to the example.

The first box at the top is a MIDI note receiver. Its three outputs are the note number, the velocity and the channel. These outputs are linked to the three last input of the *pack* box to build a list of them. The pack box receives also a message from a message box, which contains two words. The First is a kind of command which will be interpreted by the *packOSC* box, and the second is the path of the OSC message.

There are a strange link between *notein* and the message box that can explained by the fact that all the box have one "*hot input*" and possibly one or more "*cold input*". Only the "*hot input*" refresh the output, so in this program when a MIDI note arrive the message box doesn't interpret it as a number but just as a trigger. It send the message and the *pack* box build the list.

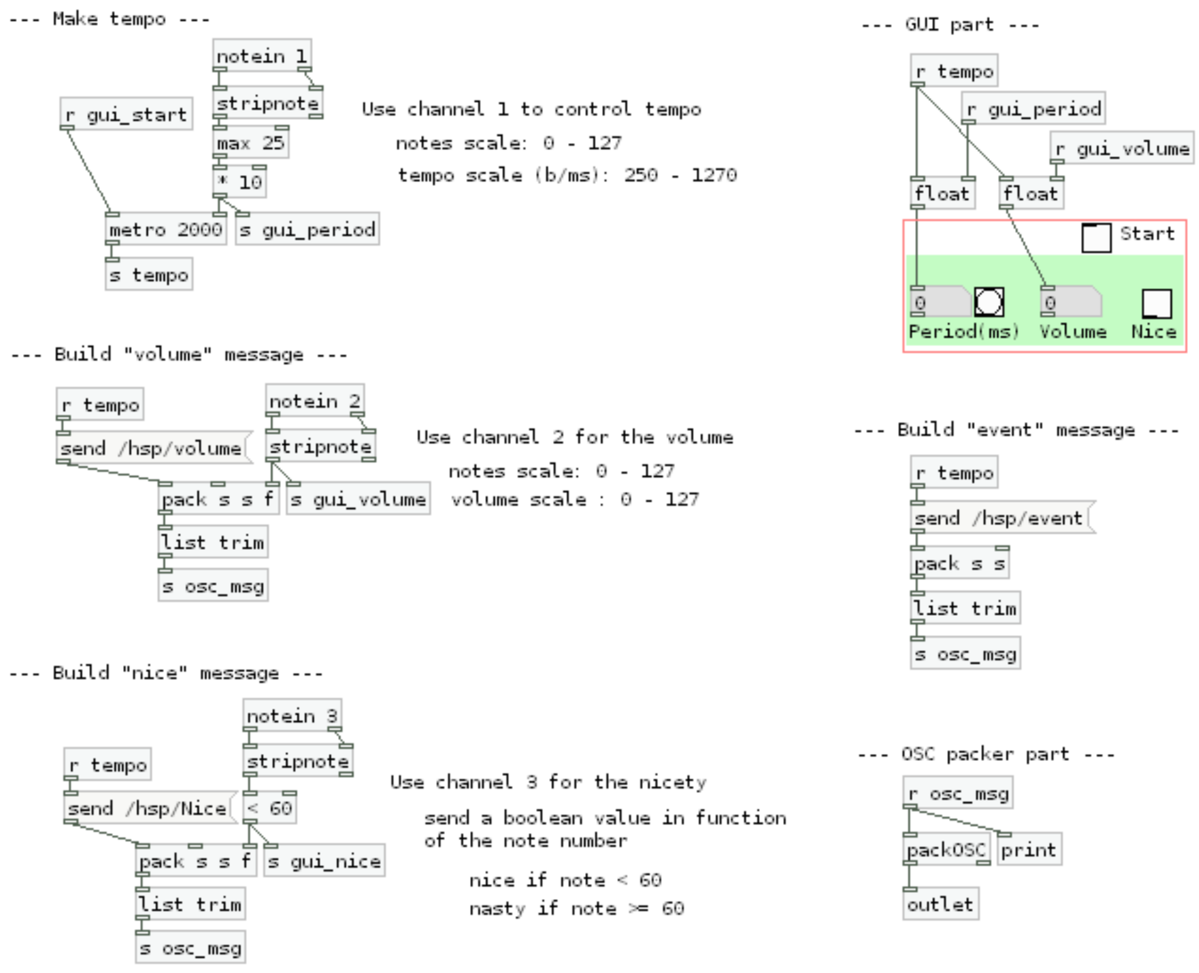
The *list* box is only used to delete the 'list' indicator before build the OSC message with *packOSC*. The next box is an output, used to link this program to another box.



```
<--- 47 409 105 100 105 0 0 0
     0 0 0 0 63 0 0 0 108 0 0 0 2
```

Picture 8. Example of Pd outlet

This other version of the translator is according with the protocol.



Picture 9. Pd program, MIDI to OSC translator

There are some difference between this and the other one. The biggest is the massive use of send and receive box for compartmentalize the program (I used alias for *send* and *receive* which are '*s*' and '*r*'). One other big difference is a GUI part (in the red square). It is use to modify the aspect of the box.

The three part on the left contain a *notein* box with an argument to limit the received notes to one channel. The first part use channel one to control a metronome, and send it as 'tempo' to trigger other parts. It also receives a start information from the GUI. The second and third parts use channel 2 and channel 3 to send volume and nicety to the GUI each tempo signal.

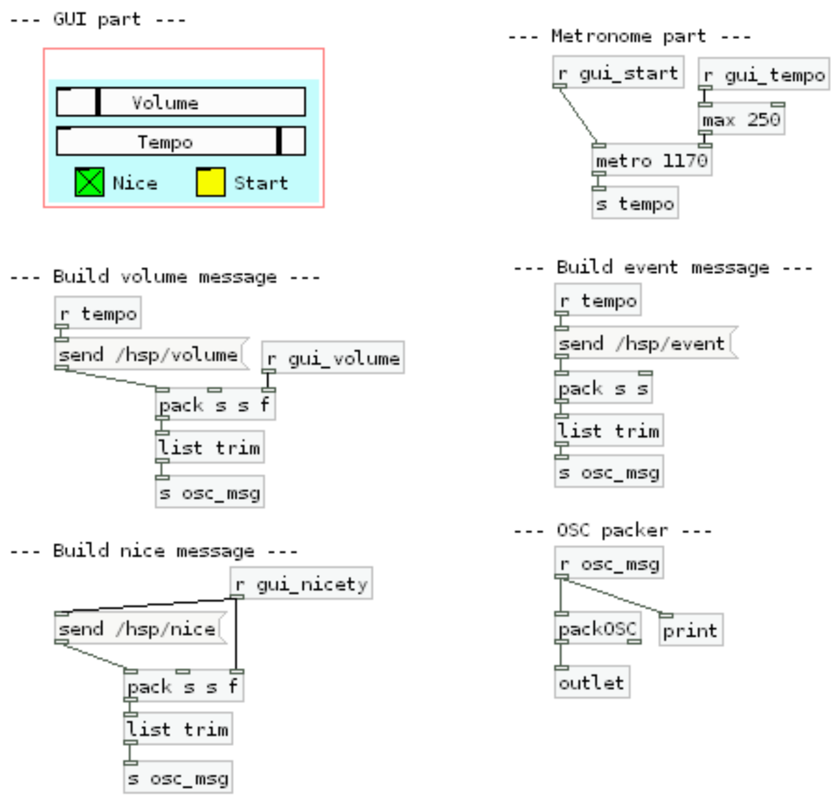
The GUI part is use to allow the user to start translation and to see what is sent. The float boxes are used to refresh the displayed values only when received a tempo signal.

The next part simply builds an 'event' message each tempo signal. And the OSC packer part makes the message ready to send.

Like the simple version after the *packOSC* box, I send messages outside.

OSC generator

It appears than Simon needs a test program to improve the new implementation of Harmony Space during its development. I chose to let the user control over what is sent, So I have wrote a program which can build OSC messages with informations from a GUI.



Picture 10. Pd program, OSC generator

It's approximately the same than the translator. I just replace all the *notein* boxes by *receive* one. The received informations come from GUI button and slide bar. For example the metronome is controlled by the box 'r gui_tempo', that is send by the GUI slide bar 'Tempo'. The fact that the slide bar sends its output signal as 'gui_tempo' is due to its internal configuration.

UDP sender

The two previous described programs build OSC message. So, now we need a program to send it. Part about the UDP function, and the graphical function that a was written

I have called this function OSC_sender but it can send all kind of UDP packet.

I was written a GUI extention of the "udpsend" Pd function.

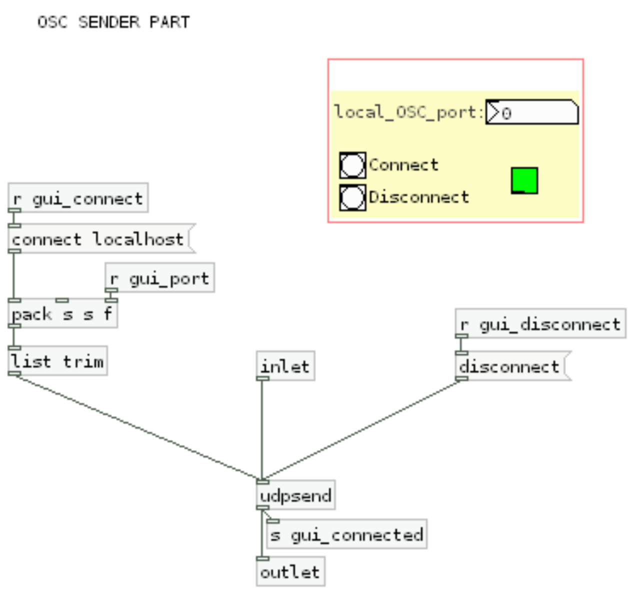
This function has one input (called "inlet" here) that is directly connected to the input of the "udpsend" box. There is also an output that I use for the "connected" GUI (button) displayer.

On the screen shot of the function you can see a red square (called canvas by Pd) that used for change the classical representation of the function (grey box with the name write in). In this square I have put two button for the connection and disconnection, a number box and a (button) displayer.

Picture 11. Pd program, UDP sender

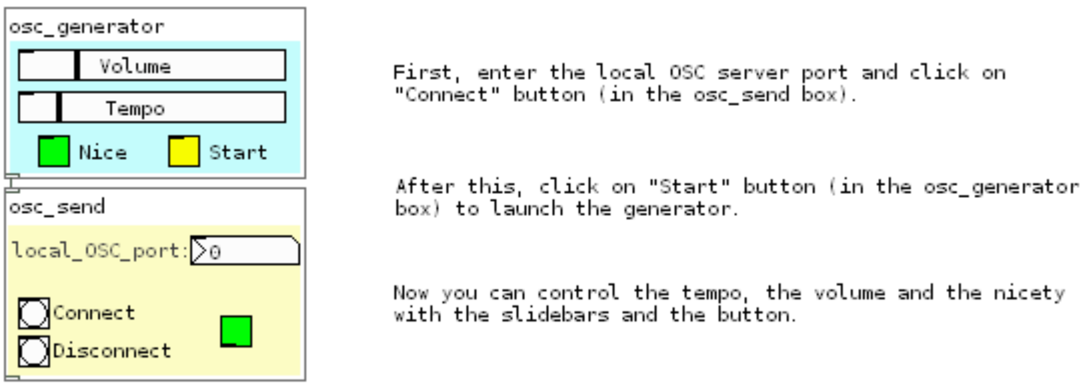
In this one there is a GUI part too, that allow the user to set the local port, Connect and Disconnect. We can see three link to the input of the udpsend box, which are a part linked to the 'Connect' button another linked to the 'Disconnect' button. And the part in the middle is the input of the box. I have named it OSCsender but it can send what you want.

Of course I know that UPD is a not connected protocol. I used this word because it's the name of the command I need to use with the *udpsend* box. I think it's a stretch of language, and the *connect* and *disconnect* command is used for open and close the socket.



Final program

In the previous part, I explained the internal programming of the boxes that I have created for the project. Now we can link the created boxes to send the Osc messages by UDP.



First, enter the local OSC server port and click on "Connect" button (in the osc_send box).

After this, click on "Start" button (in the osc_generator box) to launch the generator.

Now you can control the tempo, the volume and the nicety with the slidebars and the button.

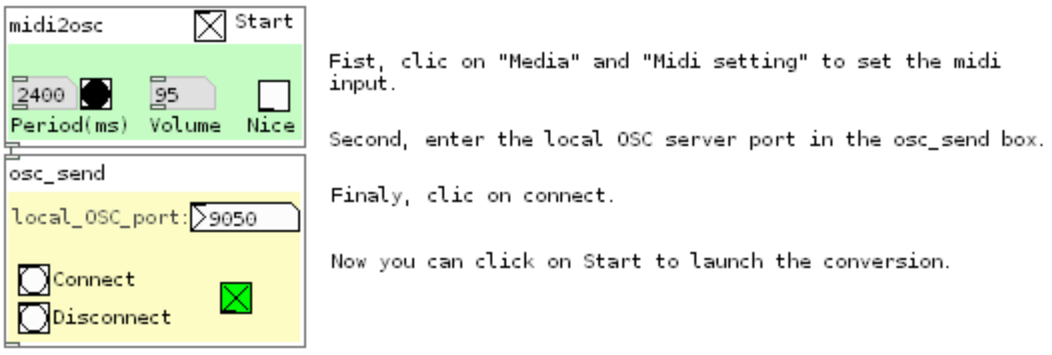
Picture 12. Pd program, final OSC generator

Picture 13. Pd program, final MIDI to OSC translator

Like I said at the beginning, when you run a Pd program, there are one window for the running program and the Pd window. So, when we run the whole project, there are four windows opened. Furthermore Pd is an programming environment, It's not user friendly. That's why I decide to programming a java version.

2. Java Version

When user run the application the graphical interface is built. In the same time a list of the MIDI devices is made. With this list the user can configure the translation, choice the input MIDI device, and for the output there are two texts fields for the host name and port. On the graphical interface there are also two buttons to launch HSP and Waveware.

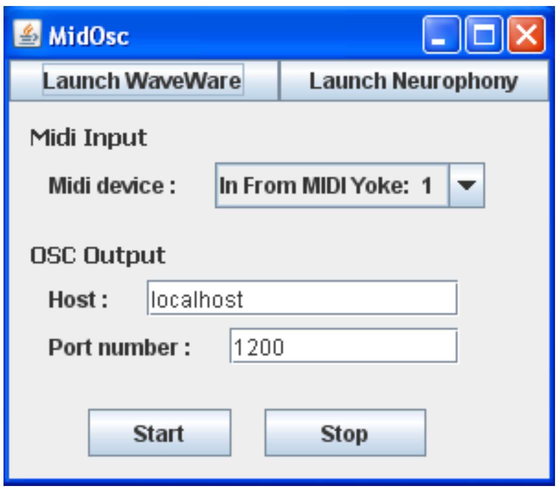


Fist, clic on "Media" and "Midi setting" to set the midi input.

Second, enter the local OSC server port in the osc_send box.

Finaly, clic on connect.

Now you can click on Start to launch the conversion.



Midi device : In From MIDI Yoke: 1

Host : localhost

Port number : 1200

Picture 14. Java MidOsc screenshot

The 'start' button launch the two most important part of the software, according to the user configuration. The first one is the MIDI receiver, its role is to receive MIDI notes and in function of the channel, send to the other part the information to build OSC messages. I was called the other part OSC engine because it is a thread with an infinite loop to send at regular interval the OSC messages. The loop time is defined by the informations from the MIDI receiver.

Classes

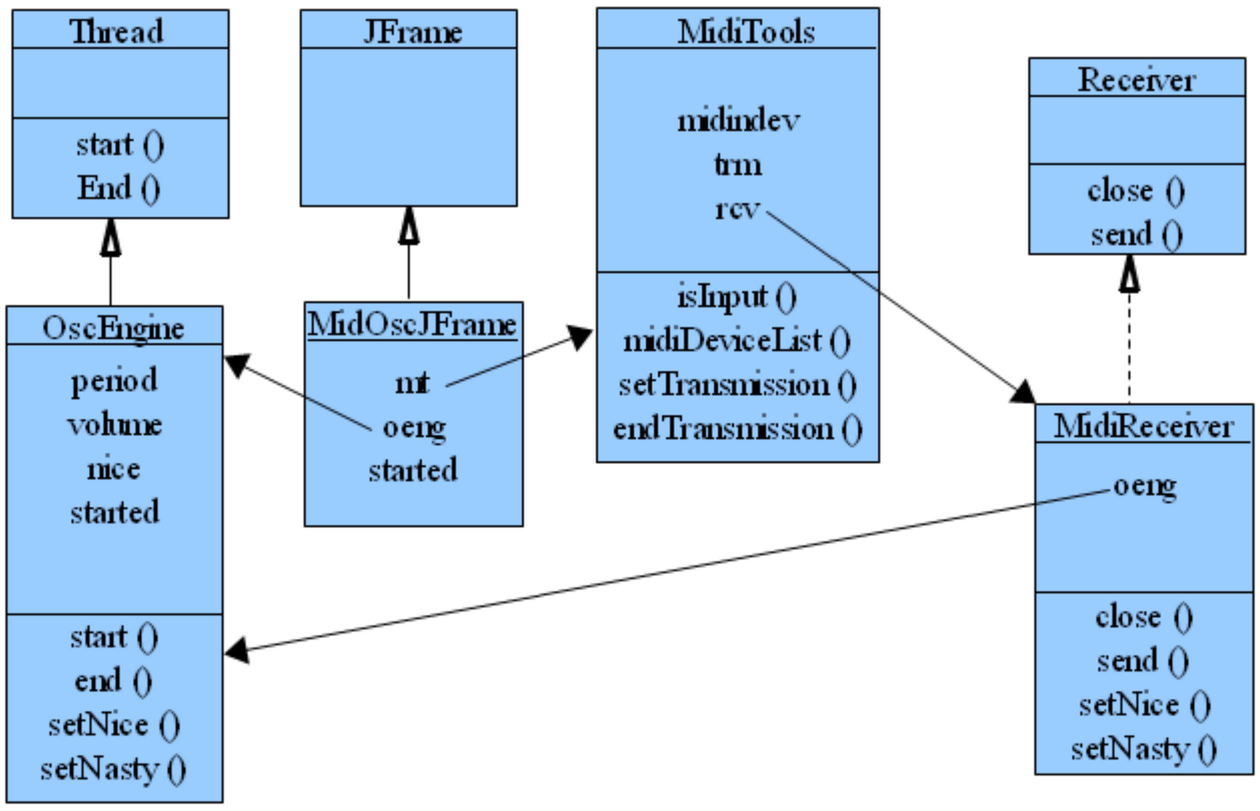


Diagram 3. UML Java program

MidiTools class is used for create the list of midi input present on the computer. The list is created by midiDeviceList method and accessible as an array of MidiDevice. It was also used for create and set the midi receiver with the method setTransmission, and to close all with endTransmission method.

The *MidiReceiver* class implement the *Receiver* interface from midi package. It need to be associate with a MIDI device, that is the role of the setTransmiter method from Miditools class. After that, the *send* method is called automatically each midi message received in the selected device. This method called some method of OscEngine to set the content of the future sent messages.

The *OscEngine* class inherit from *Thread* class. So, the *start* method launches the run function in a thread. Its role is to send the OSC message cross the socket define when calling the constructor. The metronome time is defined by MidiReceiver. The information sent is those defined in the protocol so volume, nice/nasty, and period for the next sending.

The *MidiOscJframe* class inherit from *Jframe* class. This class builds the graphical user interface, to allow the user to select the MIDI device and enter the OSC host and port. They are four buttons to start and stop the conversion, and to launch the both software (Waveware and HSP).

Future Changes

The next step is to make a more sophisticated version of the composer that perhaps need other information from the EEG. So to adapt the translator there are only two functions to modify. The *send* method of the MidiReceiver class, to change the read channel, or to interpret it in another way. And the *run* method of the OSCengine class, to change the messages sent.

CONCLUSION

The last day I have testing the whole project, that is the chain: WaveRider – Wareware – MidOsc – Neurophony. Waveware crash often, Perhaps due to the battery level of waverider during the test. And before I have tested the chain with a recorded brain wave and that works fine.

The uses of Java instead of Pd, was a good one because, I add function like the launcher and because that be more easy to use, because Java (contrary to Pd) is already installed on most computers.

Results being obtained pleasant, we can realize a more complex version.

Glossary

UBO : Université de Bretagne Occidental (University of Western Brittany)

IUP : Institut Universitaire Professionnalisé (Professionalized Academic Institute)

ICCMR : Interdisciplinary Centre for Computer Music Research

EEG : Electroencephalogram (or Electroencephalography)

OSC : Open Sound Control

BCI : Brain-Computer Interface

BCMI : Brain-Computer Music Interface

HSP : Harmony Space

Pd: PureData

Annexes

MidOscJFrame.java
MidiTools.java
MidiReceiver.java
OscEngine.java

MidOscJFrame.java

```
package midosc;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author fleon
 */
public class MidOscJFrame extends javax.swing.JFrame {

    /** Creates new form MidOscJFrame */
    public MidOscJFrame() {
        initComponents();
        init();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanelLauncher = new javax.swing.JPanel();
        jButtonWaveWare = new javax.swing.JButton();
        jButtonHSP = new javax.swing.JButton();
        jPanelConf = new javax.swing.JPanel();
        jLabelMidiIn = new javax.swing.JLabel();
        jLabelMidiDevice = new javax.swing.JLabel();
        jComboBoxMidiDev = new javax.swing.JComboBox();
        jLabelOscOut = new javax.swing.JLabel();
        jLabelOscHost = new javax.swing.JLabel();
        jTextFieldOscHost = new javax.swing.JTextField();
        jLabelOscPort = new javax.swing.JLabel();
        jTextFieldOscPort = new javax.swing.JTextField();
        jPanelStarter = new javax.swing.JPanel();
        jButtonStart = new javax.swing.JButton();
        jButtonStop = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("MidOsc");

        jPanelLauncher.setPreferredSize(new java.awt.Dimension(300, 23));
        jPanelLauncher.setRequestFocusEnabled(false);
        jPanelLauncher.setLayout(new java.awt.GridLayout(1, 0));

        jButtonWaveWare.setText("Launch WaveWare");
        jButtonWaveWare.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                jButtonWaveWareMousePressed(evt);
            }
        });
        jPanelLauncher.add(jButtonWaveWare);

        jButtonHSP.setText("Launch Harmony Space");
        jButtonHSP.setActionCommand("Launch Harmony Space");
        jButtonHSP.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                jButtonHSPMousePressed(evt);
            }
        });
        jPanelLauncher.add(jButtonHSP);

        getContentPane().add(jPanelLauncher, java.awt.BorderLayout.PAGE_START);

        jLabelMidiIn.setFont(new java.awt.Font("Tahoma", 1, 12));
        jLabelMidiIn.setText("Midi Input");

        jLabelMidiDevice.setText("Midi device : ");

        jLabelOscOut.setFont(new java.awt.Font("Tahoma", 1, 12));
        jLabelOscOut.setText("OSC Output");

        jLabelOscHost.setText("Host : ");

        jTextFieldOscHost.setText("localhost");

        jLabelOscPort.setText("Port number : ");

        jTextFieldOscPort.setText("1200");

        org.jdesktop.layout.GroupLayout jPanelConfLayout = new
org.jdesktop.layout.GroupLayout(jPanelConf);
        jPanelConf.setLayout(jPanelConfLayout);
        jPanelConfLayout.setHorizontalGroup(
            jPanelConfLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanelConfLayout.createSequentialGroup()
                .addContainerGap()
                .add(jPanelConfLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jPanelConfLayout.createSequentialGroup()
                        .add(jPanelConfLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                            .add(jPanelConfLayout.createSequentialGroup()
                                .add(10, 10, 10)
                                .add(jLabelMidiDevice)
                                .add(18, 18, 18)
                                .add(jComboBoxMidiDev, 0, 123, Short.MAX_VALUE))
                            .add(jLabelMidiIn))
                        .add(67, 67, 67))
                    .add(jPanelConfLayout.createSequentialGroup()
                        .add(jLabelOscOut)
                        .addContainerGap(206, Short.MAX_VALUE))
                    .add(jPanelConfLayout.createSequentialGroup()
                        .add(10, 10, 10)
                        .add(jPanelConfLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                            .add(jPanelConfLayout.createSequentialGroup()
                                .add(jLabelOscPort)
                                .add(18, 18, 18)
                                .add(jTextFieldOscPort, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 102,
Short.MAX_VALUE))
                            .add(jPanelConfLayout.createSequentialGroup()
                                .add(jLabelOscHost)
                                .add(18, 18, 18)
                                .add(jTextFieldOscHost, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 139,
Short.MAX_VALUE)))
                        .add(81, 81, 81))))
        );
        jPanelConfLayout.setVerticalGroup(
            jPanelConfLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanelConfLayout.createSequentialGroup()
                .addContainerGap()
                .add(jLabelMidiIn)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jPanelConfLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jLabelMidiDevice)
                    .add(jComboBoxMidiDev, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                .add(18, 18, 18)
                .add(jLabelOscOut)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jPanelConfLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jLabelOscHost)
                    .add(jTextFieldOscHost, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jPanelConfLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jLabelOscPort)
                    .add(jTextFieldOscPort, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(12, Short.MAX_VALUE))
        );

        getContentPane().add(jPanelConf, java.awt.BorderLayout.CENTER);

        jPanelStarter.setPreferredSize(new java.awt.Dimension(423, 50));

        jButtonStart.setText("Start");
        jButtonStart.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                jButtonStartMousePressed(evt);
            }
        });

        jButtonStop.setText("Stop");
        jButtonStop.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                jButtonStopMousePressed(evt);
            }
        });

        org.jdesktop.layout.GroupLayout jPanelStarterLayout = new
org.jdesktop.layout.GroupLayout(jPanelStarter);
        jPanelStarter.setLayout(jPanelStarterLayout);
        jPanelStarterLayout.setHorizontalGroup(
            jPanelStarterLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanelStarterLayout.createSequentialGroup()
                .add(43, 43, 43)
                .add(jButtonStart, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .add(18, 18, 18)
                .add(jButtonStop, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 73, Short.MAX_VALUE)
                .add(99, 99, 99))
        );
        jPanelStarterLayout.setVerticalGroup(
            jPanelStarterLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(org.jdesktop.layout.GroupLayout.TRAILING, jPanelStarterLayout.createSequentialGroup()
                .addContainerGap(16, Short.MAX_VALUE)
                .add(jPanelStarterLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jButtonStart)
                    .add(jButtonStop))
                .addContainerGap())
        );

        getContentPane().add(jPanelStarter, java.awt.BorderLayout.PAGE_END);

        pack();
    }// </editor-fold>

    private void init() {
        // MIDI Tool (list midi devices)
        mt = new MidiTools ();
        try {
            mt.midiDeviceList();
            for (int i = 0 ; i < mt.midindev.length ; i++) {
                jComboBoxMidiDev.addItem(mt.midindev[i].getDeviceInfo());
            }
        } catch (Exception ex) {
            System.out.println("err1: " + ex);
        }

    }

    /*
     * Launch WaveWare
     */
    private void jButtonWaveWareMousePressed(java.awt.event.MouseEvent evt) {
        try {
            java.lang.Runtime.getRuntime().exec("cmd /c start WAVEWARE.BAT");
        } catch (Exception ex) {
            Logger.getLogger(MidOscJFrame.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /*
     * Launch Harmony Space
     */
    private void jButtonHSPMousePressed(java.awt.event.MouseEvent evt) {
        try {
            java.lang.Runtime.getRuntime().exec("cmd /c start HSP.BAT");
        } catch (IOException ex) {
            Logger.getLogger(MidOscJFrame.class.getName()).log(Level.SEVERE, null, ex);
        }

}

    /*
     * Start the translation
     */
    private void jButtonStartMousePressed(java.awt.event.MouseEvent evt) {
        try {
            if (!started)
            {
                // disable editing on combobox and textfield (for keep selected the device)
                jComboBoxMidiDev.setEnabled(false);
                jTextFieldOscPort.setEnabled(false);
                jTextFieldOscHost.setEnabled(false);
            // start osc engine
            oeng = new midosc.OscEngine(jTextFieldOscHost.getText(), jTextFieldOscPort.getText());
            oeng.start();
            // open midi device, set receiver and transmitter
            mt.setTransmission (jComboBoxMidiDev.getSelectedIndex(), oeng);
            started = true;
        }
    } catch (Exception ex) {
        System.out.println("err2: " + ex);

    }
}

/*
 *  Stop the translation
 */
private void jButtonStopMousePressed(java.awt.event.MouseEvent evt) {
    if (started)
    {
        // close midi device
        mt.endTansmission(jComboBoxMidiDev.getSelectedIndex());
        oeng.end();
        // enable editing on combobox and textfield
        jComboBoxMidiDev.setEnabled(true);
        jTextFieldOscPort.setEnabled(true);
        jTextFieldOscHost.setEnabled(true);
        started = false;
    }
}

/**
* @param args the command line arguments
*/
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new MidOscJFrame().setVisible(true);
        }
    });
}

// My varables
private MidiTools mt;
private OscEngine oeng;
private boolean started = false;
// End of My variables

// Variables declaration - do not modify
private javax.swing.JButton jButtonHSP;
private javax.swing.JButton jButtonStart;
private javax.swing.JButton jButtonStop;
private javax.swing.JButton jButtonWaveWare;
private javax.swing.JComboBox jComboBoxMidiDev;
private javax.swing.JLabel jLabelMidiDevice;
private javax.swing.JLabel jLabelMidiIn;
private javax.swing.JLabel jLabelOscHost;
private javax.swing.JLabel jLabelOscOut;
private javax.swing.JLabel jLabelOscPort;
private javax.swing.JPanel jPanelConf;
private javax.swing.JPanel jPanelLauncher;
```

```
    private javax.swing.JPanel jPanelStarter;
    private javax.swing.JTextField jTextFieldOscHost;
    private javax.swing.JTextField jTextFieldOscPort;
    // End of variables declaration

}
```

MidiReceiver.java

```
package midosc;

import          javax.sound.midi.MidiMessage;
import          javax.sound.midi.ShortMessage;
import          javax.sound.midi.Receiver;

/**
 *
 * @author iccmr
 */
public class MidiReceiver implements Receiver
{
    OscEngine oeng;

    MidiReceiver (OscEngine aOeng) {
        oeng = aOeng;
    }

    public void close () {
    }

    /*
     *  Function called each MIDI message received
     */
    public void send (MidiMessage message, long lTimeStamp)
    {

        if (message instanceof ShortMessage)
        {   ShortMessage sm = (ShortMessage)message;
            if (sm.getCommand() == 0x90) //if it's note on
            {


                // Interpret the MIDI messages
                // and send to the OSC engine the extracted informations
                switch (sm.getChannel())
                {
                    case 0 :    int period = sm.getData1();
                                oeng.setPeriod(period<25 ? 250 : period*10);
                                break;
                    case 1 :    oeng.setVolume(sm.getData1());
                                break;
                    case 2 :    if(sm.getData1() > 60)
                                    oeng.setNice();
                                else
                                    oeng.setNasty();
                                break;
                }

            }
        }
    }

}
```

MidiTools.java

```
package midosc;

import java.util.ArrayList;
import javax.sound.midi.*;
import javax.sound.midi.MidiDevice.Info;

/**
 *
 * @author iccmr
 */
public class MidiTools {

    protected MidiDevice[] midindev;
    protected Transmitter trm;
    protected Receiver       rcv;

    /**
     *  Test if it's a midi input device
     *
     * @param MidiDevice Device to check
     */
    protected static boolean isInput (MidiDevice md) {
        // if it hasn't Receiver, it's a MIDI IN port
        if ( md.getMaxReceivers()==0 )
            return true;
        else
            return false;
    }

    /**
     *  Build a list of the midi input device
     */
    protected void midiDeviceList () throws MidiUnavailableException {
        Info info[];
        info = MidiSystem.getMidiDeviceInfo();
        MidiDevice[] midev = new MidiDevice[info.length];
        ArrayList tmpList = new ArrayList<MidiDevice>();
        for (int i = 0 ; i < info.length ; i++) {
            midev[i]= MidiSystem.getMidiDevice(info[i]);
            if (isInput(midev[i]))
                tmpList.add(midev[i]);
        }
        midindev = (MidiDevice[]) tmpList.toArray(new MidiDevice[0]);
    }

    /**
     * Set the Transmitter and the Receiver associated
     *
     * @param devNo MIDI device number
     * @param oeng to associate MidiReceiver and OscEngine
     */
    protected void setTransmission (int devNo, OscEngine oeng) throws MidiUnavailableException
    {
        midindev[devNo].open();
        trm = midindev[devNo].getTransmitter();
        rcv = new MidiReceiver(oeng);
        trm.setReceiver(rcv);
    }

    /**
     * Close the MIDI Device
     *
     * @param devNo MIDI device number
     */
    protected void endTansmission(int devNo)
    {   // close midi device
                                        midindev[devNo].close();
        trm.close();
        rcv.close();
    }

}
```

OscEngine.java

```
package midosc;

import java.net.InetAddress;
import midosc.osc.OSCMessage;
import midosc.osc.OSCPortOut;

/**
 *
 * @author iccmr
 */
public class OscEngine extends Thread {

    protected int period = 2000;
    protected int volume = 0;
    protected boolean nice = true;
    protected boolean started = true;
    final Object obj = new Object();

    OSCPortOut sender;

    OscEngine (String ahost, String aport) {
        try {
            // --- create socket
            int port = Integer.parseInt(aport);
            InetAddress host = InetAddress.getByName(ahost);
            sender = new OSCPortOut(host, port);
        } catch (Exception ex) {
            System.out.println("OscEngine: " + ex);
        }
    }


    /*
     * send periodicaly OSC message according to the value of
     * the variables: period, volume and nice
     */
    @Override
    public void run () {
```

```
    // for sendimg only when change
    int volumeTmp = 0;
    boolean niceTmp = false;

    try {

        // --- for osc message
        OSCMessage msgEvt = new OSCMessage("/hsp/event");
        OSCMessage msg;
        Object arg[] = new Object[1];

        while(started)
        {
            // --- send nice message
            if (nice != niceTmp)
            {
                niceTmp = nice;
                if (nice == true)  arg[0] = new Integer(1);
                else arg[0] = new Integer(0);

                msg = new OSCMessage("/hsp/nice", arg);
                sender.send(msg);
                System.out.println("OscEngine: send: Nice/Nasty");
            }
            // --- send volume message
            if (volume != volumeTmp)
            {
                volumeTmp = volume;
                arg[0] = new Integer(volumeTmp);
                msg = new OSCMessage("/hsp/volume", arg);
                sender.send(msg);
                System.out.println("OscEngine: send: volume(" + volumeTmp + ")");
            }

            // --- send event message
            sender.send(msgEvt);
            System.out.println("OscEngine: send: event");

            // to execute action in the loop periodically
            synchronized (obj) {
                    obj.wait(period);
            }
        }

    } catch (Exception e) { System.out.println("oscengne: " + e);}
}


protected void setVolume (int newVolume)
{   volume = newVolume;
}

protected void setPeriod (int newPeriod)
{   period = newPeriod;
}

protected void setNice ()  { nice = true; }

protected void setNasty ()  { nice = false; }
```

```
    /**
     *  go out of the loop
     */
    protected void end ()
    {
        started = false;
        synchronized (obj) {
                obj.notify();
        }
    }
}
```