Porting Max Patches to the Smart Controller

Angelo Fraietta
University of Western Sydney
PO Box 859
Hamilton NSW, 2303
Australia
angelo_f@smartcontroller.com.au

Abstract

This paper will examine some of the differences and similarities between Max and the Smart Controller languages by examining the composition 'Whirling Wheels,' originally written by the author for Max in 1997, which was ported to and performed on the Smart Controller.

Introduction

Many composers have developed gesture based interactive instruments and responsive environments by mapping input data collected using control voltage to MIDI converters, which is input to a computer running an algorithmic compositional program such as Max (Paine 2001, ; Winkler 1995, ; Rowe 1993, ; Winkler 1998). One of the advantages of using Max is that composers are able to use graphical or iconic representations instead of having to type in text commands, one advantage being that a graphical representation may be more easily comprehended than with text alone (Favreau et al. 1986, ; Burt 1999, ; Rowe 2001, ; Puckette 1991). Max has become so popular, that it is used in many music departments across the globe, with more musicians being able to program in Max than in the C++ programming language (Rowe 2001).

During the years that these composers have been using Max, many have built up libraries of patches for performing specific functions, enabling them to reuse these patches, which in turn, enables them to build new patches quickly (Winkler 1998). Although the Smart Controller integrates the control voltage and algorithmic composition into a single hardware unit, which can remove the requirement for a personal computer at an installation or performance, composers might assume this advantage may not offset the effort required in learning the graphical language of the Smart Controller. The task of learning how to program the Smart Controller can be facilitated by porting existing compositions already written in Max, for performance in the Smart Controller.

This paper will examine some of the differences and similarities between Max and the Smart Controller languages by examining the composition *Whirling Wheels,* originally written by the author for Max in 1997, which was ported to and performed on the Smart Controller.

Whirling Wheels

Whirling Wheels is a quadraphonic piece where three distinct sounds were made to appear as though they were moving about the room. The impression of rotating sound was effected by sending MIDI notes and pan messages to an FM synthesiser, allocating two MIDI channels per voice, and assigning each of those two channels to a separate group output of the synthesiser. For example, the voices in channels 1, 3 and 5 are assigned to group output 1, while channels 2, 4 and 6 are assigned to group output 2. The result is two stereo pairs, one from each group output. The left and right of each pair are assigned to diagonal quadrants, with group output 1 being mapped to front-left and back-right speakers; while group output 2 is mapped to front-right and back-left as shown in *Figure 6*.

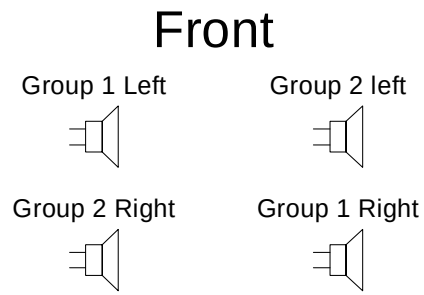

Figure 6 MIDI channel to speaker mapping

This is accomplished using MIDI by repeating notes in the channel pairs with the same note number, but with velocity values ninety degrees out of phase. When the velocity of a channel reaches zero, the panning on the channel toggles to either 0 or 127. To make a sound rotate through each speaker in a circle, the repeated sound must appear in two speakers, with the intensity in each speaker according to position. Intensity of sound is realised by using the velocity of the note. To explain the rotation of sound from front-left full circle in a clockwise direction, we will assume that MIDI channel 1 left is at the front left speaker, MIDI channel 2 left is at front right, MIDI channel 1 right is at back right, and MIDI channel 2 right is at back-right.

Sound appears at the front-left speaker, therefore the panning of MIDI channel 1 must be fully

left with sound generated with a note velocity of 127; MIDI channel 2 is panned fully left with sound produced with a noted velocity of 0. As sound approached the front- right speaker, the velocity of notes generated on MIDI channel 1 decrease, reducing sound from front-left speaker; while the velocity of the notes on MIDI channel 2 increase, increasing its intensity of sound in the front right speaker.

When the sound is at the front-right speaker, notes from MIDI channel 2 have a velocity of 127, while notes from MIDI channel 1 have a velocity of zero. MIDI channel 1 has its pan changed to fully right, thus assigning sound output to the back-right speaker; its velocity, however, is zero, which means that the pan change is not audible at that instant.

As sound moves from the front-right speaker to the back-right speaker, the velocity of the notes on MIDI channel 2 decrease, while the note velocity increases in MIDI channel 1. When the note velocity in MIDI channel 1 reaches 127 and the note velocity in MIDI channel 2 reaches 0, the sound is fully at the back-left speaker. MIDI channel 2 changes pan to fully right, and thus assigning its sound to the back-left speaker.

Sound approaches the back left speaker as notes on MIDI channel 2 increase in velocity, while those on MIDI channel 1 decrease in velocity. When the note velocity on MIDI channel 1 becomes 0 and that on MIDI channel 2 becomes 127, the sound is fully in the back-left speaker. MIDI channel 1 changes pan to fully left, thus assigning its sound to the front left speaker.

When note velocities on MIDI channel 1 reach 127, note velocities on MIDI channel 2 are 0, MIDI channel 2 is then panned fully left. This has been a full revolution.

MIDI channel pairs 3&4 and 5&6 work exactly the same, with odd MIDI channels numbers being assigned to group 1, while the even MIDI channels are assigned to group 2. In addition to turning a voice on or off, the performer is able to modify the volume, speed of rotation, and pitch of each voice. This piece was originally realised in Max, with performance control performed using sliders, toggles, and message boxes as the performance controls

Max Patch Algorithm

The Max patch was developed as three distinct patcher objects, one for each pair of MIDI channels, each with controls connected to the inlets. A single channel pair is shown in *figure 7*. The voice rotation is started by setting the toggle labelled 'rotate,' which sends a one value into the fifth inlet of the patcher. The speed of rotation is adjusted by the slider marked 'metro speed,' which is input to the sixth inlet of the patcher through a number box. The volume of the voice is controlled by the slider marked 'vol,' which is input to the seventh inlet of the patcher.

The vertical slider on the left sets the pitch of the note, and is input to the first inlet of patcher layer 1.

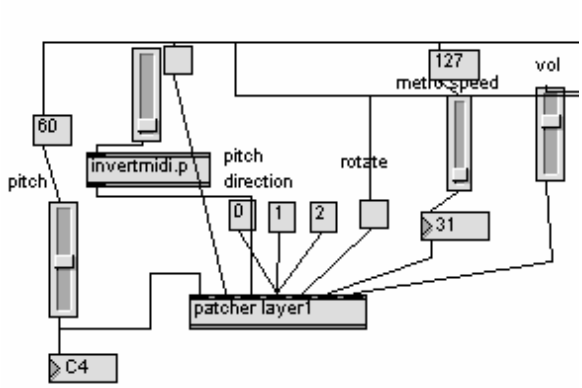

Figure 7 Top Level Max Patch

It is also possible to have the pitch change automatically by setting the toggle, going into the second inlet of the patcher, which starts a counter. The rate of this change is controlled by the slider, which goes into the third inlet of the patcher (through a scaling patch 'invertmidi.p'), while the direction of the pitch change counter is determined by the message boxes going into fourth inlet.

Smart Controller Translation

Within the Smart Controller, it was not necessary to have slider controls and toggles for each channel because control was effected through *AnalogIn* and *DigitalIn* objects, which received the control voltages from the external instrument. Also each voice channel was encapsulated into separate patch files, which are equivalent to Max patch objects. Within the Max patch, initialisation was performed by the user pressing a 'bang' object. Within the Smart Controller, initialisation is performed when the first digital input message received, causing the *FlipFlop* to generate an output, causing the trigger to generate a tick[3] message, causing the three *MessageStore* objects to send an initialisation number into the associated patch inlets, shown in *figure 8*. If the DigitalIn object sends another message, the *FlipFlop* will not produce an output because it is already in its set state; similar to using a 'onebang' object in Max.

[3] A tick message is identical to the Max bang message.

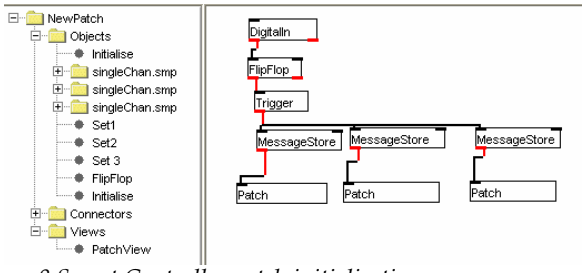

Figure 8 Smart Controller patch initialisation

The included patch file, shown in *figure 9* as two separate performance views, contains an initialisation algorithm in addition to the connection of analogue and digital inputs to another sub patch. The initialisation algorithm, shown in the performance view on the bottom of *figure 9*, mapped the associated digital and analogue inputs to the number input to its inlet by using *Calculate* objects to set the input digital and analogue channels. The performance view on the top of *figure 9* shows the mapped analogue and digital inputs as inputs to another patch-file that is functionally similar to the 'patcher layer 1' in the Max patch originally shown in *figure 7*.

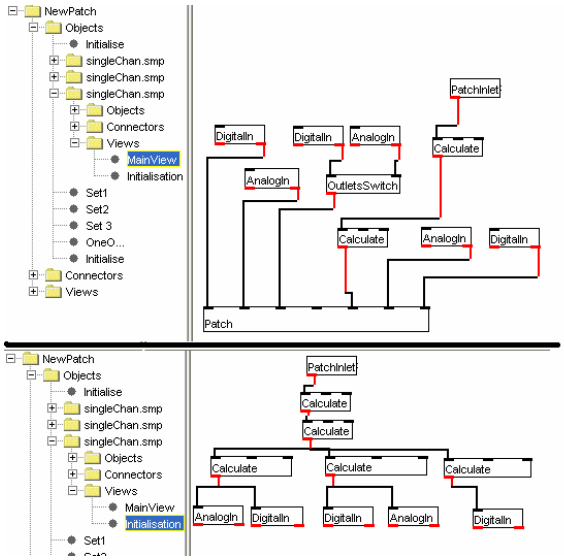

Figure 9 Mapping digital and analogue inputs inside each sub patch

Within the Max version of the piece, the ninety degree phase difference between each channel pair was effected through the use of a 0 to 127 counter and a table that generated numbers ninety degrees out of phase to the index, as shown in *figure 10*.

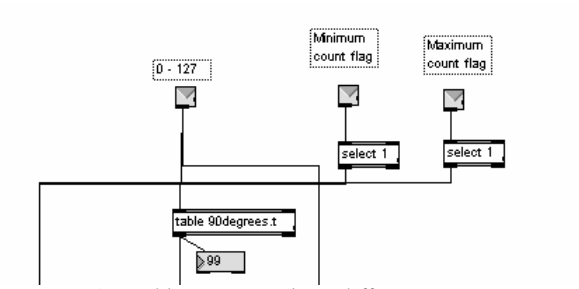

Figure 10 Using table to create phase difference

Within the Smart Controller, the phase difference was achieved through the use of two up-down counters set to count ninety degrees out of phase by setting the reset count of the leading counter to start at 63, while the lagging counter starts at 0. These two counters are shown as the circled objects within two different performance views in *figure 11*.

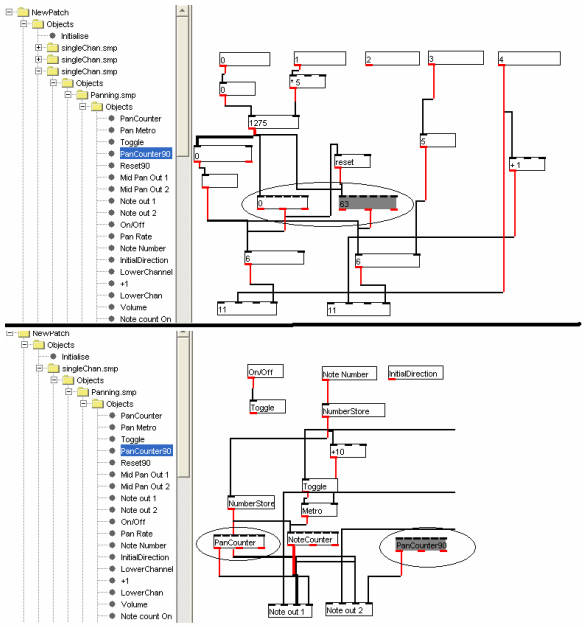

Figure 11 Counters ninety degrees out of phase

The output of these two counters has two functions, shown in the different performance views in *figure 11*. In the bottom performance view, the count output is used as the data 2 input to a MidiOut object configured to generate a note on message. This results in notes being generated with velocities ranging from 0 to 127.

In the top performance view, the underflow outlet of the counter is sent to a sub-patch to toggle the panning on the channel. Each time a counter reaches 0, the sub patch toggles from 0 to 127, which is sent to the data2 inlet of a MidiIn object configured to generate a MIDI controller 10 (pan) message, causing the MIDI channel to be hard left or right.

Object Placement

Within Max, the positioning of an object on the screen affects the algorithm because objects on the right are called before objects on the left. In Max, this was overcome by using delay objects with a value of 1 in order to make an object on the left receive a message before an object on the right. This was not necessary within the Smart Controller because it is possible to set and change the order at which connections will be executed from an outlet. Additionally, repositioning of PatchInletPorts or Patch OutletPorts—equivalent to moving inlets and outlets in a patcher in Max—does not effect the outlet number. For example, repositioning of the two circled PatchInletPorts shown at the bottom of *figure 12* to different positions on the display would have no effect on its interface to the higher level patch connected to those inlets, shown in the top of *figure 12*.

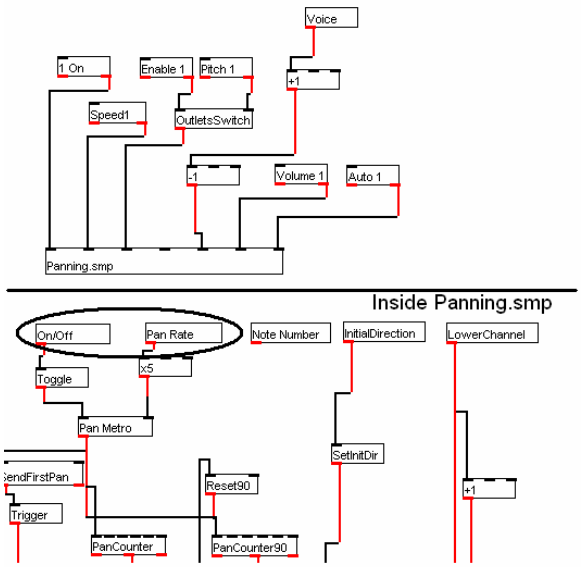

Figure 12 Inlet position in patches

Object Differences

Many of the object types using in the Smart Controller can be mapped as direct replacements of Max types. In many cases, however, Smart Controller objects have configuration options, which in turn reduces the number of different object types required. For example, where one might use a 'ctlin 12 1' object in Max to receive controller information data for MIDI controller 12 on MIDI channel 1; Smart Controller would use a *MidiIn* object configured to filter MIDI controller 12 messages on channel 1, as shown in *figure 13*.

Figure 13 Configuring MidiIn object

Table 1 provides a conversion list, loosely mapping Smart Controller object types to Max object types.

Smart Controller Type	Similar Max Types	Comments on differences
Analogue In	Nil	
Analogue Out	Nil	
Calculate	Arithmetic objects	The operator is defined as an attribute and can also be changed through an inlet. Can also be made to perform calculation through the right inlet.
Counter	Counter	The counter can have differing initial direction, count step size, and initial count.
Delay	Delay	
Digital In	Nil	
Digital Out	Nil	
Display	print	
Flip-flop	Combination of one-bang and toggle	Generated output is 1 for non-zero input to left inlet and 0 for non-zero input to right inlet. Right inlet may also cause output generation.
Inlets Switch	Switch	

Smart Controller Type	Similar Max Types	Comments on differences
Message Store	Message box	
Metro	Metro	
Midi In	Combination of MIDI input object types	MIDI message types can be filtered so as to only allow certain messages that meet criteria to generate output.
Midi Out	Combination of MIDI output object types	Can also be configured to turn off a previous note if generated by this object
Number Store	Number Box	
OSC In	Third party external	
OSC Out	Third party external	
Outlets Switch	Gate	
Patch	Patcher	
Patch From File	Patch	A patch from file can be converted to an internal patch and vice versa
Patch Inlet Port	Inlet	Changing the position on the display does not effect the inlet number
Patch Outlet Port	Outlet	Changing the position on the display does not effect the outlet number
Random Gen.	Random	
Selector	Select	
Sequencer	Seq	The MIDI bytes and meta data are sent to different outlets. Playback speed can be changed while sequence is playing
Table	Table	
Toggle	Toggle	
Trigger	Bang	

Table 1 Smart Controller to Max conversion

Conclusion

Learning to program the Smart Controller can be expedited by porting compositions originally written for Max. Spoken languages are often taught by porting standard dialogue from a known language into the language being learnt. The same can be accomplished in music programming by porting algorithms from a known language to a new one. This, in turn, reduces the load placed upon the composer in that they would not be required to input as much creative energy into learning the new language; instead, the process could be seen as arranging an existing work for a new instrument.

References

Burt, Warren. 1999. "An Email Interview with Warren Burt." Interview by G. Schiemer. Chroma: Newsletter of the Australasian Computer Music Association 25:2-6. <http://www.acma.asn.au/Chroma25.pdf>.

Favreau, Emmanuel, Michel Fingerhut, Olivier Koechlin, Patrick Potacsek, Miller Puckette, and Robert Rowe. 1986. Software developments for the 4X real-time system. In proceedings of the International Computer Music Conference. Royal Conservatory of Music, Den Haag, Netherlands.

Paine, Garth. 2001. Interactive sound works in public exhibition spaces: an artists perspective. In proceedings of Waveform 2001: the Australasian Computer Music Conference. University of Western Sydney.

Puckette, Miller. 1991. "FTS: a real-time monitor for multiprocessor music synthesis." Computer Music Journal 15 (3):58-67.

Rowe, Robert. 1993. Interactive music systems : machine listening and composing. Cambridge, Mass.: MIT Press.

———. 2001. Machine musicianship. Cambridge, Mass. ; London: MIT Press.

Winkler, Todd. 1995. Making motion musical: Gesture mapping strategies for interactive computer music. In proceedings of International Computer Music Conference. Banff, AB, Canada.

———. 1998. Composing interactive music: techniques and ideas using Max. Cambridge, Mass.: MIT Press.