The Smart Controller – shifting performance boundaries

Angelo Fraietta

PO Box 859, Hamilton NSW, 2303
email: angelo_f@bigpond.com

Abstract

Many composers today are using control voltage to MIDI converters and laptop computers running algorithmic composition software to create interactive instruments and responsive environments. Using an integrated device that combines the two devices at the performance would reduce latency, improve system stability, and reduce setup complexity. Composers and performers, however, have chosen not to use an integrated device due the boundaries imposed upon them by the available devices.

Users were forced to program their patches using assembler. Secondly, it was difficult for users to upgrade the firmware inside their device. Users were also unable to build and modify the firmware in the way MAX users were able to create new types of objects.

This paper examines these issues and explains how the Smart Controller overcame these boundaries. Additionally, examples are given where composers are now using the Smart Controller in their works in preference to laptop computers.

1 Introduction

Many composers today are using algorithmic compositional techniques that map certain physical and conceptual gestures to musical parameters (Doornbusch 2002). Gesture based interactive instruments and responsive environments (Paine 2001) are often developed using MIDI controllers—such as the ICube, SensorLab, and MIBURI MIDI jump suit – using these as inputs to laptop computers running algorithmic compositional and performance software (Mustard 2002).

There can, however, be problems encountered using a laptop computer at the actual performance venue. These problems include latency, complexity in cabling and setup, system stability, and excessive wear-and-tear on expensive computer equipment (Kartadinata 2002). In many cases, the performance does not require a sophisticated graphical environment; rather, the laptop merely executes the required algorithm for the performance. In such a situation, an integrated device that combines the control voltage and MIDI input and output (hereafter I/O), as well as running the algorithm itself, into a single unit is preferable because it would solve many of the problems stated.

There have, however, been restrictions placed upon composers, performers, and hardware developers when considering an integrated device. Firstly, users have been forced to program their patches for these devices using assembler or another text based language. Secondly, the software became static within the device. The firmware was locked into the hardware device, making it difficult for users to upgrade the firmware. This in turn gave users a feeling of limitation shortly after developing some facility with the device (Riddell 2001). Furthermore, users were unable to build and modify the firmware due to licensing restrictions, the cost of real-time operating systems (hereafter RTOS), and the requirement for cross-compilers. Finally, the hardware platform itself became a boundary, whereby software developed originally for a Motorola system, for example, would not be easily ported to another chip type.

Most of these problems can be addressed and solved by not only determining what resources are required for the composer, performer and developer; but also at what point in time the resource is required.

During development of the Smart Controller, these issues were identified, addressed and implemented in an open system. Additionally, examples are given where composers are now using the Smart Controller in their works in preference to laptop computers.

2 Performance

There are integrated devices, such as the MIDI Tool Box (Schiemer 1998; Bandt 2001), that integrate control voltage and MIDI into a single unit. This device has many advantages in performance over a non-integrated system.

The first advantage is that the time between a performer making a musical gesture and the time that sound is generated is reduced significantly. This statement is not significant until one considers that the processing power of the microcontroller within these integrated devices is miniscule when compared to the processing power available in laptop computers today (Ganssle 2002). The problem is twofold.

Firstly, in the non-integrated system, control voltages are converted to MIDI before entering the laptop computer. To perform the conversion from control voltage to MIDI, the device sequentially scans a number of inputs, compares the value with the previously read value, generates a MIDI message based upon that input, and then continues the scanning process (Gayman 1988). To calculate the amount of time taken from input to output, one must add the time taken to read the input, the time taken to

calculate what MIDI message must be sent, and the time taken to generate the entire MIDI message. This value must then be multiplied by the number of inputs the device has. For example, the Infusions Systems I-Cube has maximum sampling rate of 250Hz when using only one input and generating a MIDI control message. This amounts to a worst case latency of four milliseconds. This latency increases to over fifteen milliseconds when using all thirty-two inputs (Infusion Systems 1998).

Next, the laptop computer must now read the MIDI at its input, perform the required computation, and generate the required output. Although laptop computers have an enormous amount of processing power, much of these resources are spent by preemptive operating system that must service devices such as displays, hard disks, communications ports, and resident programs. Consequently, the computing power made available to the algorithmic performance program may not be scheduled immediately, which in turn introduces latency (Messick 1998). The resources serviced by the operating system are often not required for the performance. This is effectively a waste of processing power. This problem showed itself when code running on a 40MHz 386 computer, running the Real Time Executive for Multiprocessor Systems (hereafter RTEMS) RTOS, executed identical C plus-plus code faster than a 1.133GHz Athlon machine running Windows 2000 (Fraietta 2002).

The second advantage of the integrated setup is system stability. Laptop computer systems rely on the underlying computer operating system that can crash or freeze at any moment. By comparison, the embedded system can be made to boot immediately into the appropriate algorithm, and then in the event of a program malfunction, reboot the machine (Murphy and Barr 2001). In many cases, a sound installation may be required to run for days or weeks (Coburn 2002). Many programs for PC or Macintosh computers are designed to run for only a few hours at a time, whereas some embedded devices are expected to run for years before cycling power (Ganssle 2002).

The final advantage of the integrated device in a performance situation is the physical constraints. "Weight and size isn't everything when it comes to stage (or touring) fitness. The plastic casing and the display on hinges sure don't encourage one to drop the PB [Power Book] from a table" (Kartadinata 2000). Furthermore, the multitude of cables required in a setup considerably increases the complexity and encroaches upon the performance space itself (Kartadinata 2002). An integrated device can reduce the number of cables required, and subsequently, the number of connections required during performance setup.

These points lead to the question as to why composers and performers are not replacing their laptop computers with integrated devices. The answer is linked to the way composers are creating their patches.

3 Graphical patch programming

The main disadvantage for composers using an integrated device is the lack of a graphical programming environment to create and run patches. The learning curve for these devices can be too great for many composers as they are required to program in assembler or another structured text language (Burt 1999). Many musicians want to move quicker than they are able to, which in many cases leaves the composer dissatisfied with the result of their work (Arveiller 1982). This is because there is an enormous quantity of programming involved before one can make any music (Burt 1999).

Graphical programming environments are not limited to artists, but rather, have become a standard inclusion within engineering circles (Douglass 2001, 1999; Mintchell 1999, 1998). The main advantage of graphical representation is that the comprehension of the structured algorithm, in many cases, can be more easily facilitated than with text alone (Favreau et al. 1986). A large percentage of the brain is involved in the processing of visual information, which is a large motivation for visually or graphically representing information (Tufte 2001).

This is the case today also with music software, where people are moving to software packages that use a graphical or iconic representation of the algorithm (Burt 1999). The laptop computer has the advantage over the embedded system in that it has the processing power to run software packages with graphical interfaces.

These music software packages, such as MAX, Audio Mulch, PD, and Algorithmic Composer, enable the composer to create their algorithms by connecting iconic objects together within a graphical environment. An object is joined to another object by connecting an outlet from one object to the inlet of another object (Puckette 1988), similar to the way older analogue synthesisers used patch cables to connect modules together (Pressing 1992). This method of programming has become extremely popular within the computer music community today; consequently, more musicians are able to program using MAX than are able to in C plus-plus (Rowe 2001).

Implementing a graphical programming environment requires additional processing power – a very limited commodity on embedded systems (Ganssle 2002). This effectively means that the algorithm composition must be performed on a desktop or laptop computer. Although computer aided software engineering tools enable one to generate code using a graphical environment (Larman 2002), the resultant code does not allow one to run the program using the icons that are used to generate the code. One is therefore not able to manipulate their patch within the graphical environment.

This problem can be solved when one considers that composition and performance often take place at different times and in different places. The graphical interface is only required during composition, therefore justifying the use of the laptop computer at that point in time. During the performance, however, the graphical interface is not required, thus making way for the embedded device. Engineers using programmable controllers have been moving toward this paradigm since Modicon developed the first programmable controller for General Motors in 1969 (Gayman 1988).

The Smart Controller solves this problem by separating the graphical interface from the underlying scheduling algorithm, known as the engine, using a stack system based loosely on the ISO/OSI seven-layer model (Lemieux 2000). The patch editor contains the graphical interface and calls travel down the stack to communicate with the underlying data structures in the engine through the bottom layer. In the case of the simulator, a single function to a shared library using a pair of character buffers effects all intercommunication. In the case of an embedded hardware device, this lower layer is a communications port or a MIDI data stream. The source code for the engine in both the simulator and the physical embedded device is identical, including the scheduling algorithm, with the exception of operating system primitives and hardware drivers. This effectively enables the composer to program and rehearse the algorithm almost completely using a graphical interface. Additionally, the composer can test the behaviour without ever having the embedded hardware.

4 Flexibility and scalability

A perceived advantage of the laptop computer is that software can be upgraded or modified quite easily, which allows the software developer to revise and post software updates using the Internet (Riddell 2001). This, however, is not an exclusive feature for laptop systems. Embedded systems programmers have overcome this by using flash memory and a bootstrap loader that enables the device to download a new binary executable from an I/O port and then boot into the new code (Glass 1998). The Smart Controller has implemented this feature, and subsequently a user can download new executable binaries for their hardware as the Smart Controller system matures and grows in sophistication.

Another advantage with the laptop system is that users in some cases are able to enhance the algorithmic software in some way. For example, MAX users can create and use their own object types using MAX externals, which in turn enables users to share the newly created objects with other users of MAX – one of the features that helped make MAX so popular today (Rowe 2001). This is not normally possible with hardware devices because the whole of the firmware must be re-compiled into a single executable binary file. However, due to the recent advent of the free software and open-source movement (Gatliff 2000), particularly the use of GNU compilers (Gatliff 2000) and RTEMS, the Smart Controller has overcome this boundary also.

RTEMS is a free open-source RTOS that was originally developed for the US Army, but was released for non-military uses later (Sherrill 2003). When asked why RTEMS was given away instead of being sold, Mark Johannes of On-line Applications Research (hereafter OAR) stated “It was a natural fit to release RTEMS as open source to leverage the environment that promotes the further development and utilization of the product” (Johannes 2001). A great advantage of using RTEMS in the Smart Controller is that all the development tools are free, the operating software is free, and there is free support available also. The new Smart Controller firmware can be built on a Windows, Linux, or OSX machine at no cost to the developer (Norum 2003). RTEMS has already been used as the operating system in an assortment of projects ranging from avionics and space research through to robotics and system on chip audio players (Sherrill 2003; Azuara and Kiatisevi 2002).

RTEMS runs on more than forty hardware platforms (Sherrill 2003), and subsequently, the Smart Controller as an embedded device could be ported quite easily to other hardware platforms.

5 Uses in performance

The Smart Controller was first used in performance at the University of Western Sydney in September 2003 to perform *Whirling Wheels* - a quadraphonic performance where three distinct sounds appeared to rotate through four speakers (Fraietta 2002). The piece was performed by controlling a guitar shaped device that generated control voltages. The Smart Controller performed an algorithm that generated MIDI note and controller messages, sending them to a Roland SY-99 FM synthesizer. The impression of rotating around the room was achieved my by sending note velocities and pan control values at high speed to the SY-99.

Also, the Smart Controller is being used in a sound installation where the public interact with small metallic bells. The Smart Controller senses changes in the environment, controls solenoids to mechanically strike the bells, and schedules the different phases within the performance (Norman 2003).

6 Conclusion

The Smart Controller enables composers to generate patches from the comfort of their iBook or PC workstation, while at the same time providing efficient, effective, and reliable performance in concert that is only obtainable through a purpose-built embedded device. This has been achieved by recognising what resources were required and at what

time. In the case of a week long installation, composers do not have to leave their laptop computers public places. Additionally, it enables composers to use a graphical patching paradigm they are already familiar with. Also, composers are able to effectively compose, simulate, and run patches separate from the hardware device using a Windows or Macintosh OSX computer. The firmware can be upgraded easily and can even be modified by developers. Additionally, the software can be ported to other hardware platforms should the need or desire arise. Finally, the Smart Controller is successfully being used by composers today.

7 Acknowledgments

I would like to thank first and foremost the greatest artist and engineer – the creator of all things seen and unseen – Jesus Christ. Many of the concepts used in developing the Smart Controller have been already used by Him in His creation.

Additionally, I would thank Joel Sherrill, Ralf Corsepius, Eric Norum, and all the people in the RTEMS community. Without their help and dedication, the Smart Controller would never become a reality.

Finally, I would like to thank Dr Jim Franklin for his encouragement and the University of Western Sydney for making the required funding available.

References

Arveiller, Jacques. 1982. "Comments on university instruction in computer music composition." *Computer Music Journal* 6 (2):72-78.

Azuara, Luis, and Pattara Kiatisevi. 2002. Design of an audio player as a system-on-a-chip. Masters thesis, Institute of computer science, University of Stuttgart, Stuttgart.

Bandt, Ros. 2001. *Sound sculpture : intersections in sound and sculpture in Australian artworks*. St. Leonards, N.S.W.: Craftsman House.

Burt, Warren. "An Email Interview with Warren Burt." Interview by Greg Schiemer. *Chroma* 25 (1999): 3-6.

Coburn, Robert. 2002. Composing space: the integration of music, time, and space in multi-dimensional sound installations. In proceedings of *Form, space, time: the Australasian Computer Music Conference*. Royal Melbourne Institute of Technology.

Doornbusch, Paul. 2002. The application of mapping in composition and design. In proceedings of *Form, space, time: the Australasian Computer Music Conference.* Royal Melbourne Institute of Technology.

Douglass, Bruce Powel. 1999. "UML statecharts." *Embedded Systems Programming* 12 (1):22-42.

———. 2001. "Capturing real-time requirements." *Embedded Systems Programming* 14 (12):18-30.

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.

Fraietta, Angelo. 2002. Smart Controller -- Artist Talk. In proceedings of *Form, space, time: the Australasian Computer Music Conference*. Royal Melbourne Institute of Technology.

———. 2002. Whirling Wheels. Sydney: Angelo Fraietta.

Ganssle, Jack G. 2002. "Breaking into embedded." *Embedded Systems Programming* 15 (9):43-46.

Gatliff, Bill. 2000. "Embedding with GNU: the GNU compiler and linker." *Embedded Systems Programming* 13 (2):66-78.

———. 2000. "Open source is already delivering." *Embedded Systems Programming* 13 (10):88-93.

Gayman, David J. 1988. "An old favorite gets new standards." *Manufacturing Engineering* 100 (1):55-59.

Glass, Brett. 1998. "There in a Flash: Flash Memory for Embedded Systems." *Embedded Systems Programming* 11 (1):81-88.

Infusion Systems. 1998. *I-Cube System Manual*. Last accessed 5 March 2003. http://www.infusionsystems.com/support/icubex-111-manual.pdf.

Johannes, Mark. 2001. Email to A. Fraietta, Re: History of RTEMS, 7 March 2001.

Kartadinata, Sukandar. 2000. *hardMax*. Last accessed 23 October 2000. http://members.xoom.com/Sukandar/hardMAX.html.

———. 2002. *The Gluiph -a platform for integrated electronic musical instruments*. Last accessed 18 February 2003. http://www.glui.de/proj/gluiph.html.

Larman, Craig. 2002. *Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process*. 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR.

Lemieux, Joseph. 2000. "The OSEK/VDXStandard: Operating System and Communication." *Embedded Systems Programming* 13 (3):90-108.

Messick, Paul. 1998. *Maximum MIDI : music applications in C++*. Greenwich: Manning.

Mintchell, Gary A. 1998. "PCs power programming tools." *Control Engineering* 45 (12):42-50.

———. 1999. "Graphic interfaces are programmer's friends." *Control Engineering* 46 (11):57-65.

Murphy, Niall, and Michael Barr. 2001. "Watchdog timers." *Embedded Systems Programming* 14 (11):79-80.

Mustard, Johnathan. 2002. Correlating movement in space to the parameters of sound. In proceedings of *Form, space, time: the Australasian Computer Music Conference*. Royal Melbourne Institute of Technology.

Norman, Anne. 2003. *Power Pole Bells*. Anne Norman. Last accessed 12 March 2003. http://home.vicnet.net.au/~amncrow/PPBells.html.

Norum, W. Eric. 2003. *Getting started with EPICS on RTEMS*. Last accessed 26 February 2003. http://www.aps.anl.gov/epics/modules/base/RTEMS/tutorial/tutorial.html.

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.

Pressing, Jeff. 1992. *Synthesizer performance and real-time techniques*. Madison, Wis.: A-R Editions.

Puckette, Miller. 1988. The Patcher. In proceedings of *the International Computer Music Conference*. GMIMIK, Kologne, Germany.

Riddell, Alistair. 2001. Email to A. C. M. A. m. l. acma-l@list.waikato.ac.nz, Re: acma-l Midi Controllers, 10 October 2001.

Rowe, Robert. 2001. *Machine musicianship.* Cambridge, Mass. ; London: MIT Press.

Schiemer, Gregory. 1998. MIDI Tool Box an interactive system for music composition. Ph.D. diss., Macquarie University, Sydney.

Sherrill, Joel. 2003. Email to A. Fraietta, History of RTEMS, 7 March 2001.

———. 2003. *RTEMS Applications*. Last accessed 4 March 2003. http://www.oarcorp.com/~joel/rtems/apps.html.

———. 2003. *RTEMS Ports and BSPs*. On-line Applications Research. Last accessed 28 February 2003. http://www.oarcorp.com/RTEMS/Features/Ports___BSPs/ports___bsps.html.

Tufte, Edward R. 2001. *The visual display of quantitative information*. 2nd ed. ed. Cheshire, Conn.: Graphics Press.