# The Smart Controller Workbench

Angelo Fraietta
University of Western Sydney
PO Box 859
Hamilton NSW 2303 Australia
+61-2-49697577
angelo_f@bigpond.com

## ABSTRACT
The Smart Controller is a portable hardware device that responds to input control voltage, OSC, and MIDI messages; producing output control voltage, OSC, and MIDI messages (depending upon the loaded custom patch). The Smart Controller is a stand alone device; a powerful, reliable, and compact instrument capable of reducing the number of electronic modules required in a live performance or installation, particularly the requirement of a laptop computer. More powerful, however, is the Smart Controller Workbench, a complete interactive development environment. In addition to enabling the composer to create and debug their patches, the Smart Controller Workbench accurately simulates the behaviour of the hardware, and functions as an in-circuit debugger that enables the performer to remotely monitor, modify, and tune patches running in an installation without the requirement of stopping or interrupting the live performance.

## Keywords
Control Voltage, Open Sound Control, Algorithmic Composition, MIDI, Sound Installations, programmable logic control, synthesizers, electronic music, Sensors, Actuators, Interaction.

## 1. INTRODUCTION
Many composers today are creating gesture based interactive instruments and responsive environments, predicated on the detachment of the excitation and sonification [24], enabling composers to map physical and conceptual gestures to musical parameters [27]. In many cases, the instruments are implemented in three stages [25]: a sensing stage, whereby data is collected using transducers or sensor devices that detect change in the physical environment; a processing stage for mapping or manipulation, whereby the data is applied to algorithms or patches; and a response stage, where the manipulated data is sent for sound generation to a synthesizer, such as on-board DSP or a MIDI synthesizer; or an electro-mechanical instrument, such as LEMUR's musical robots [4] and the Interactive Bell Garden [7, 19]. The demarcation between the successive stages can be reduced by combining two or more of the stages on a single machine, as is currently done in programs like MAX/MSP [25] and PD [22].

The Smart Controller is a portable hardware device that can combine all three stages into a single unit. The Smart Controller responds to input control voltage (hereafter CV), Open Sound Control (hereafter OSC) [28], and MIDI messages; producing output CV, OSC, and MIDI messages (depending upon the loaded custom patch). The power of the Smart Controller, however, is not primarily based upon its processing power; rather it is founded upon the ease and degree of control that composers have in being able to develop and refine algorithms for the device using the Smart Controller workbench, a software API and simulator.

## 2. HISTORY
The idea of the Smart Controller was seeded when the author collaborated with Guy Robinson in the development of the *Virtual Drum Kit*, which entailed a drummer playing an imaginary drum kit [6]. The instrument required a CV to MIDI converter and a computer running Max [23] to decode and remap the MIDI. The author dreamed about the possibility of one day integrating the control voltage input and the data filtering and algorithmic components into a single small hardware device. This dream became a reality in the Smart Controller due to the inspiration of two products that on the surface appear unrelated: a music program for algorithmic composition and a hardware device used to monitor and control water pump and sewage stations. The Smart Controller was therefore inspired in two parts: as a software package and as a hardware device.

### 2.1 Software Inspiration
The software component of the Smart Controller was originally inspired by the Max programming language [23] while the author was still an undergraduate. Max was only available for the Apple Macintosh platform at that time; the author, however, did not own a Macintosh. Subsequently, he developed Algorithmic Composer [5] for Microsoft Windows so he could continue to create algorithmic music using the patching paradigm used by Max.

Algorithmic Composer was written with the view that it could be later ported to other platforms and operating systems—this underlying code became the domain logic code of the Smart Controller [6].

### 2.2 Hardware Inspiration
The hardware concept for the Smart Controller was inspired by a program called ISaGRAF, "a control software environment that enables the creation of local or distributed control systems" [11, 12]. The author was first introduced to this environment when developing firmware for Serek Controls (formerly Hunter

Watertech); where ISaGRAF was used to program, control, and monitor programmable logic controllers used for controlling water pump stations through telemetry [3, 14].

One of the graphical languages ISaGRAF uses is called "function block diagram" and is like a set of interconnected graphical objects [16], a software equivalent to connecting electronic chips with virtual wires [21]. This method of programming is the same paradigm used by iconic music programs such as Max and PD.

An impressive feature of ISaGRAF is that an engineer can develop the algorithm for the programmable logic on a desktop computer, simulate the hardware within the development environment, and then download the algorithm to the hardware device through a communications port or network. More notable, however, was the ability for an engineer to connect to the embedded device through the network and monitor the internal variables within the development environment, almost as if the algorithm was being run on the local desktop computer.

## 3. SMART CONTROLLER SYSTEM

The Smart Controller system consists of a physical hardware device and a software workbench that performs the functions of patch editor, simulator, and hardware debugger.

### 3.1 Hardware Device

The hardware device comes in differing levels of functionality, enabling a person to add to their hardware configuration as their requirements change. The first level is the *Dumb Controller,* which is simply a MIDI to CV / CV to MIDI converter, and has been used by various artists who have chosen to use a personal computer for the processing stage of their performance [1, 2, 10, 17, 20]. The next level is the *Dumb OSC Converter,* CV to OSC / OSC to CV converter, which is similar to the Dumb Controller except that it also performs OSC conversion. The final level, the *Smart Controller,* integrates the processing stage inside the hardware, removing the requirement for a personal computer for the processing stage.

All versions have up to sixteen of each of the following CV inputs and outputs: 0-5VDC ten-bit analogue inputs, switched digital inputs, 0-5VDC digital outputs, and 0-10VDC eight-bit analogue outputs. The analogue inputs are scanned once every six milliseconds while the digital inputs are scanned once every millisecond. The devices have one MIDI input and two independent MIDI output ports.

Within the later two versions, the number of CV and MIDI inputs and outputs can be doubled through the addition of another I/O board, resulting in thirty-two of each type of CV, two MIDI inputs and four MIDI outputs. These devices also have an Ethernet port and can communicate using OSC; reading OSC messages from ten different UDP ports simultaneously and transmitting OSC on any number of ports.

Instead of basing the Smart Controller upon hardware and specific compilers, the system is based upon the patching and connection algorithm originally developed in Algorithmic Composer and later refined and optimized for use within an embedded system. It is portable to a number of different hardware platforms and operating systems as demonstrated in the simulator implementation for the Windows and Mac OSX operating systems [8]. The hardware device uses the Real Time Executive for Multiprocessor Systems (hereafter RTEMS), an RTOS originally developed for the US Army Missile Command. RTEMS currently runs on more than forty different hardware platforms [8], which means that the Smart Controller can be easily ported with RTEMS as the underlying RTOS to many other hardware platforms as they become available or necessary. It would be possible to add DSP to the Smart Controller later using PD, which has already been shown to work in an embedded system [13], while providing a mechanism whereby objects can be manipulated and accessed externally [15].

### 3.2 Workbench

The workbench is a software program that is used to program, simulate, and debug the hardware device.

#### *3.2.1 Patch Editor*

The patch editor is a graphical interface used by the composer to create and modify patches. The workbench consists of two components: the graphical user interface (hereafter GUI), which determines how the patches are displayed on the user's screen; and the Smart Controller engine, which is the underlying domain logic. The workbench can operate in simulation mode, where the objects are created within the same process as the workbench; or in monitor mode, where the GUI displays the objects that are on the embedded device. The underlying logic of the patch is not contained within the graphical environment; instead, the graphical environment makes calls to the underlying Smart Controller engine. This gives the composer or performer the flexibility to monitor and adjust the visual representation without affecting the logical structure.

While the workbench is in simulation mode, the patches are created within the same process space as the GUI. This would be similar to starting MAX and creating a patch—the patch actually exists on the computer that has the GUI. When the workbench is operating in monitor mode, the GUI makes calls to the hardware through the communications link—such as an Ethernet or RS232 port—displaying the information in the same way as would be done in simulation mode.

In order to make the GUI identical for as many platforms as possible, the distributed workbench for Windows and OSX is written and distributed as a JAVA package, with the underlying engine written and compiled in ANSI C++ [26]. The GUI communicates with the engine using the Java Native Interface, which enables Java code to call functions written in native compiled languages such as C and C++ [18].

The patch editor provides three primary graphical representations of the patch: performance view, tree view, and edit view.

##### *3.2.1.1 Performance View*

The performance view enables the composer to view part of or all of a patch in a diagrammatic format, representing the objects as icons, and connections as lines between the objects. Objects are easily connected together when in performance view. Inlets are represented at the top of the icons, while outlets are on the bottom. This is very similar to the representation used by Max and PD.

Within the performance view, it is possible to alter the text that is inside each icon so as to display the name or the current default attribute value of the objects, as shown in figure 1.

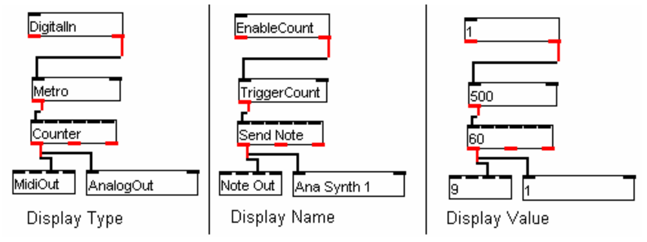
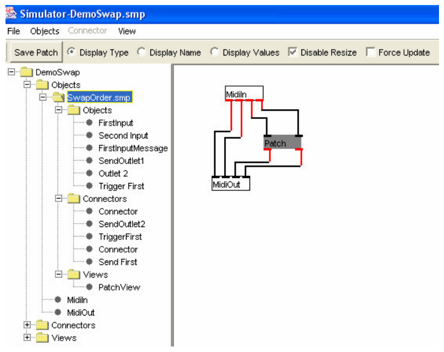


**Figure 1 Smart Controller performance view**

### *3.2.1.2 Tree View*

The tree structure displays all objects, connectors, and views within the context of the entire patch structure. Each patch or sub-patch is represented in the tree with three branches: Objects, Connectors, and Views. The objects, connectors, and views have the name of the object in text next to the icon. The Objects branch displays all objects that are within the first level or layer of abstraction within the patch relative to the Objects branch. If an object within the branch is a patch, it also has three branches, which displays the next level of encapsulation or abstraction. Figure 2 displays the tree view of an object that contains a sub-patch.[1]

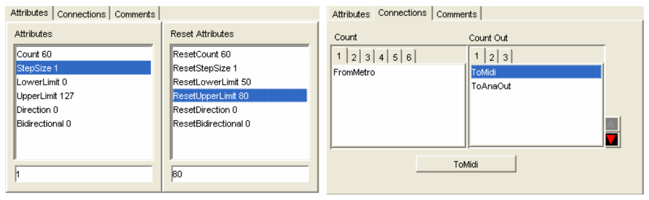


```
DemoSwap
  Objects
    SwapOrder.smp
      Objects
        FirstInput
        Second Input
        FirstInputMessage
        SendOutlet1
        Outlet 2
        Trigger First
      Connectors
        Connector
        SendOutlet2
        TriggerFirst
        Connector
        Send First
      Views
        PatchView
    MidiIn
    MidiOut
  Connectors
  Views
```

**Figure 2 Tree view displaying objects within a sub-patch**

### *3.2.1.3 Edit View*

The edit view allows manipulation of an object's internal data. In the same way that an object has parts inherited from a common ancestor and parts that are unique to the object type, the edit window has different pages that enable access to the appropriate data. The pages are separated as Attributes, Connections, and Comments, indicating the type of information accessed by the page.



Attributes

Count 60
StepSize 1
LowerLimit 0
UpperLimit 127
Direction 0
Bidirectional 0

1

Reset Attributes

ResetCount 60
ResetStepSize 1
ResetLowerLimit 50
ResetUpperLimit 80
ResetDirection 0
ResetBidirectional 0

80

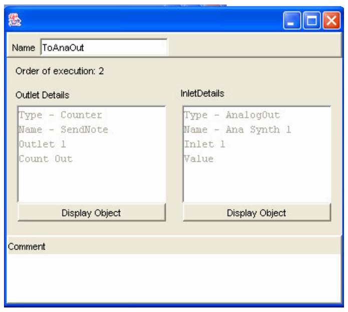
FromMetro

ToMidi
ToAnaOut

**Figure 3 Object attributes and connections pages**

The object attributes page, shown on the left of figure 3, allows the user to display and modify the internal attributes of the object. The connections page, shown on the right of figure 3, allows the user to display information about all the connections to the object and allows the user to alter the order in which the connectors are called in the outlets.

The left side of the connections page displays the inlets, while the right displays the outlets. The inlet or outlet number can be selected by selecting the appropriate tab at the top of the window, which in turn displays the connectors that are attached to the object through the selected inlet or outlet. Selecting the connector causes the connector to become highlighted in the performance view. The details about the connector can be found by displaying the connector edit view, shown in figure 4.



Name ToAnaOut

Order of execution: 2

Outlet Details

Type - Counter
Name - SendNote
Outlet 1
Count Out

InletDetails

Type - AnalogOut
Name - Ana Synth 1
Inlet 1
Value

Comment

**Figure 4 Connector edit view**

The connector edit view displays the two objects and associated inlets and outlets joined by the connector. It is possible to display the edit view of either or both of the objects, which in turn causes the selected object to become highlighted within the performance view, making it possible to navigate between the objects and connectors using the object and connector edit views.

### *3.2.1.4 Alternative Patch Editors*

Although the Java version of the patch editor is extremely detailed, the patch editor can be modified by third party developers to create a completely different interface because the source code is available online [9]. It is also possible to develop a patch editor in other languages or environments—such as Visual C++, Visual Basic, Delphi, Code Warrior, Cocoa, and Xcode—because the Smart Controller engine exists in the underlying shared libraries.

### *3.2.2 Simulator*

The workbench can be used as an alternative to algorithmic software packages such as Max or PD. When in simulation mode, OSC and MIDI messages are communicated through the Ethernet and MIDI ports of the computer running the simulator, which means that a composer or performer can use the simulator as a performance tool without ever acquiring the Smart Controller hardware device. This occurred at the Newcastle Electrofringe festival[2] where the Smart Controller simulator was used to demonstrate how OSC can be used to transmit messages between different computers.

CV inputs and outputs can be simulated on the workbench by displaying the Smart Controller simulator input and output windows. The composer can simulate an analogue input by manipulating sliders, or a digital input by selecting a checkbox. Simulated CV output is displayed as a slider for analogue output or as a small panel for digital output. This effectively enables a

[1] A sub-patch is equivalent to a patch containing a patcher object in Max.

[2] http://www.electrofringe.org/

composer to simulate an entire installation performance before obtaining the Smart Controller hardware.

### *3.2.3 Debugger*

It is possible within the workbench to connect to and monitor any number of Smart Controller hardware devices on a network. After detecting a Smart Controller, the patch inside the Smart Controller can be read back into the workbench, allowing the user to utilise the advanced features of the patch editor. Furthermore, it is possible to create and modify patches within the embedded device without stopping the existing patch or performance.

## 4. CONCLUSION

The Smart Controller Workbench provides a high level interactive development environment that enables composers to effectively develop, test, and modify their electro-acoustic and electro-mechanical instruments and installations in real-time. It also allows real-time patching to take place, altering the algorithm during performance without stopping the existing program.

## 5. ACKNOWLEDGMENTS

With respect to the OSC implementation, I would like to thank the Australia Council for providing funding, and Chris Johns of Cybertec Pty. Ltd. (http://www.cybertec.com.au/) for developing the CS8900A device driver for RTEMS.

I acknowledge the Apple University Consortium (AUC) for their provision of an Apple Computer.

This project has been assisted by the Commonwealth Government through the Australia Council, its arts funding and advisory body.

I would like to thank Dr Garth Paine for his encouragement and support as my Ph.D. supervisor.

## 6. REFERENCES

[1] Brown, A. *Australian Digital Instrument Building*. in *Converging Technologies: the Australasian Computer Music Conference*. 2003. Western Australian Academy of Performing Arts, Edith Cowan University: Australasian Computer Music Association.

[2] Clemen, H. *Interfaces for Public Use Interactive Installations: Some Design Concepts, Problems and Possible Solutions*. in *Converging Technologies: the Australasian Computer Music Conference*. 2003. Western Australian Academy of Performing Arts, Edith Cowan University: Australasian Computer Music Association.

[3] Entus, M., *Running lift stations via telemetry.* Water Engineering & Management, 1989. **136**(11): p. 41-43.

[4] Eric Singer, et al. *LEMUR's Musical Robots*. in *International Conference on New Interfaces for Musical Expression (NIME)*. 2004. Shizuoka University of Art and Culture, Hamamatsu, Japan.

[5] Fraietta, A., *Algorithmic Composer*. 1998, University of Western Sydney.

[6] Fraietta, A. *Smart budgeting for a Smart Controller*. in *Waveform 2001: the Australasian Computer Music Conference*. 2001. University of Western Sydney: Australasian Computer Music Association.

[7] Fraietta, A. *Incremental sound installation development using the Smart Controller*. in *Converging Technologies: the Australasian Computer Music Conference*. 2003. Western Australian Academy of Performing Arts, Edith Cowan University: Australasian Computer Music Association.

[8] Fraietta, A. *The Smart Controller - shifting performance boundaries*. in *Boundaryless music: the International Computer Music Conference*. 2003. National University of Singapore: International Computer Music Association.

[9] Fraietta, A., *Smart Controller Workbench Source Code*. 2004.

[10] Hewitt, D. and I. Stevenson. *Emic-Extended Mic-stand Interface Controller*. in *Conference on New Musical Interfaces for Music Expression (NIME-2003)*. 2003. Montreal.

[11] ICS Triplex plc, *ICS Triplex ISaGRAF Inc. First in open control*. 2004.

[12] Jeffreys, J.R., *Structured programming tools for embedded systems.* Control Engineering, 1998. **45**(12): p. 200.

[13] Kartadinata, S. *The Gluiph:a nucleus for Integrated Instruments*. in *Conference on New Interfaces for Musical Expression (NIME-03)*. 2003. Montreal.

[14] Langnau, L., *A step closer to easier PLC programming.* Material Handling Engineering, 1995. **50**(13): p. 23.

[15] Martin Kaltenbrunner, Günter Geiger, and S. Jordà. *Dynamic Patches for Live Musical Performance*. in *International Conference on New Interfaces for Musical Expression (NIME)*. 2004. Shizuoka University of Art and Culture, Hamamatsu, Japan.

[16] Mintchell, G.A., *Graphic interfaces are programmer's friends.* Control Engineering, 1999. **46**(11): p. 57-65.

[17] Monro, G., *Red Grains*. 2003.

[18] Naughton, P. and H. Schildt, *Java 2: the complete reference.* 3rd ed. 1999, Berkeley: Osborne/McGraw-Hill.

[19] Norman, A., *Power Pole Bells and the Bell Garden.* Sounds Unlimited: building the instruments: Sounds Australian -- Journal of the Australian Music Centre, 2003(62): p. 38-39.

[20] Opie, T. *Granular synthesis: experiments in live performance*. in *Form, space, time: the Australasian Computer Music Conference*. 2002. Royal Melbourne Institute of Technology: Australasian Computer Music Association.

[21] Pollard, J., *PLCopen, IEC 61131-3 address motion integration.* Control Engineering, 2001. **48**(7).

[22] Puckette, M., *Pure Data*.

[23] Puckette, M. and D. Zicarelli, *Max--An Interactive Graphical Programming Environment*. 1990, Opcode Systems: Menlo Park.

[24] Roads, C., *The computer music tutorial*. 1996, Cambridge, Mass.: MIT Press. xx, 1234.

[25] Rowe, R., *Interactive music systems : machine listening and composing*. 1993, Cambridge, Mass.: MIT Press. x, 278.

[26] Stroustrup, B., *The C++ programming language*. 2nd ed. 1991, Reading, Mass. ; Sydney: Addison-Wesley. xi, 669.

[27] Winkler, T. *Making motion musical: Gesture mapping strategies for interactive computer music*. in *International Computer Music Conference*. 1995. Banff, AB, Canada: The International Computer Music Association.

[28] Wright, M. and A. Freed. *Open Sound Control: State of the Art 2003*. in *International Conference on New Interfaces for Musical Expression (NIME)*. 2003. Montreal, Quebec, Canada.