Audio Engineering Society
Convention Paper

Presented at the 121st Convention
2006 October 5–8 San Francisco, CA, USA

EuCon: An Object-Oriented Protocol for Connecting Control Surfaces to Software Applications

Steve Milne[1], Phil Campbell[2], Scott Freshour[1], Rob Boyer[1], Jim McTigue[1], and Martin Kloiber[1]

[1] Euphonix, Inc., 220 Portage Ave, Palo Alto, CA, 94306, USA
smilne@euphonix.com

[2] Hobbyhorse Music LLC, PO Box 60533, Palo Alto CA 94306-0533
phil@philcampbell.com

ABSTRACT

This paper describes a control surface to application protocol that addresses the problem of raising user interface efficiency in increasingly complex software applications. Compared with existing MIDI based protocols, this protocol was designed to have enough bandwidth, high control resolution, and wide variety of controls to provide software application users with the rich and efficient experience offered by modern large format mixing consoles. Recognizing that today's audio engineer uses many different applications, it is able to simultaneously control multiple applications running on one or more computers from a single control surface. To give users the widest possible choice of applications, object-oriented design was utilized to promote ease of adoption by software developers.

1. INTRODUCTION

Software applications used in audio and video production are becoming increasingly complex, incorporating the functions previously performed using hardware mixers, editors, recorders, and effects processing units traditionally found in a studio. The highly evolved, efficient, and familiar hardware user interfaces composed of knobs, faders, switches, and indicators have been replaced by the software application's mouse and keyboard-based Graphical User Interface (GUI). Many software applications, such as Digital Audio Workstations (DAWs), mimic hardware controls in their GUIs, but the mouse and keyboard limit the speed at which an operator can change these controls. The mouse's single click and drag gestures are slow compared with the simultaneous control of several knobs, faders, and switches that two hands and ten fingers allow. Also, the size of the computer screen used by software applications limits the number of controls that can be seen and accessed without scrolling. This puts GUI interfaces at a speed disadvantage when compared to a big array of hardware controls such as those found on a large format mixing console.

EuCon is a control surface to software application protocol designed to address these problems. It can support user interfaces with thousands of high-resolution controls. This results in a user experience that combines the advantages of large physical control surfaces with the flexibility and power of software applications.

2. BACKGROUND

Recognizing the ergonomic advantages hardware control surfaces have in improving user information input and processing [1], a number of control surfaces have been introduced over the years that work with software applications [2, 3]. Typically, these control surfaces are accompanied by protocols that allow communication with software applications [4]. Although intended for control of musical instruments, the OpenSound Control protocol [5] has also been used for control surface to application control [6].

As computer performance increases, DAWs are being used for tasks traditionally performed on large format mixing consoles [7]. These tasks include film mixing, TV post production, and high end music production. These projects can exceed several hundred tracks in size. Efficient workflow is a necessity. In a film mix, for example, the film director and other professionals are spending valuable time making creative judgements. Watching an engineer spend time making unnecessary gestures can result in frustration. Efficient workflow requires fast access to channel gain levels, mute switches, editing functions, and changes to equalization (EQ), dynamics, and plug-in parameters. Often several applications are used together; for example a DAW and a video editor for video post production or a multitrack DAW and stereo mastering editor for music production. Each of these applications may have their own unique user interface. Switching between the different modes of operation can further impede workflow.

Unfortunately, many control surfaces designed for use with a DAW are not well suited for these tasks. They have failed to address the increased mental workload and its impact on workflow and operator efficiency. The number of channel strips – typically between eight and 32 - doesn't allow for fast access to many channel gain levels at once in projects with large track counts. The number of knobs for plug-in editing – often one per strip or four to eight globally shared across all channels– prohibits fast changes to dynamics, EQ, and plug-in controls, which may have dozens to hundreds of controls. The number of switches for global functions, typically ranging from ten to 50, doesn't approach the hundreds of global menu functions found in a DAW.

The bandwidth limitations of the serial Musical Instrument Digital Interface (MIDI) protocol used by many control surfaces creates a tradeoff between responsiveness and the number of controls. With 31.25 Kbit/s MIDI, for example, a maximum of 35 controls can be controlled in a 33 ms video frame interval (assuming three byte MIDI commands). This falls short of the many hundreds or even thousands of controls found in a large format mixing console or large DAW project that need to be updated at least once per video frame during automation playback. Response time is important. In an experiment, Grossberg, et al, found that in complex computer tasks where there are many possible methods to obtain a solution, users adapted their approach based upon the interface response time [8].

In addition to bandwidth limitations, many of the MIDI protocols have insufficient resolution in their data types, for example 128 discrete values for a knob, compared

with over a thousand discrete values that might be necessary for accurately choosing an EQ frequency value.

None of the protocols or controllers, to the knowledge of the authors, can simultaneously control multiple applications. One recently introduced control surface [9] has a wide variety of controls, but does not give the user the choice of controlling applications from different vendors.

3. DESIGN GOALS

EuCon was designed to address the limitations mentioned in the above section. It is a protocol for communicating between control surfaces and software applications. Specifically, its goals are to:

- Allow creation of sophisticated hardware user interfaces like those found on large format mixing consoles.
- Provide high control resolution.
- Provide a wide variety of controls, while being extensible.
- Provide enough bandwidth to support thousands of controls with low latency.
- Control multiple applications simultaneously from one control surface.
- Control a single application from multiple surfaces.
- Work with as many software applications as possible, giving users the widest possible choice.
- Be easy to program, requiring little effort for software application developers to adopt.

4. OVERVIEW

EuCon uses TCP/IP [10] to communicate between control surfaces and applications. This allows for a variety of physical layers, such as Ethernet [11], IEEE-1394 [12], or USB [13]. As of this writing, all existing EuCon control surfaces use 100 Mbit/s Ethernet. This provides three orders of magnitude more bandwidth than MIDI and will scale up as Ethernet speeds increase.

EuCon is written in C++ [14], a widely used object-oriented [15] programming language. A Software Development Kit (SDK), consisting of header files, examples, and documentation, is available for distribution.

To use EuCon, a programmer models a user interface for their application using predefined software objects that represent faders, knobs, switches, and other controls. Larger objects can be built from these objects. A channel strip object, for example, could contain a fader, mute and solo switches, eight knobs, and a text display object. A mixer object would contain channel strip objects. Programming is more straightforward when software objects model real-world physical hardware.

The EuCon programmer never sees the contents of the packets transmitted over TCP/IP. They only deal with creating, destroying, and manipulating objects using the C++ Application Programming Interface (API). This makes EuCon transport independent.

EuCon maps actual physical controls on a hardware control surface to software objects, which in turn control the application. Figure 1 below shows a software object representing a knob, which controls an EQ frequency parameter in the application. EuCon has mapped it to a physical knob on the control surface.

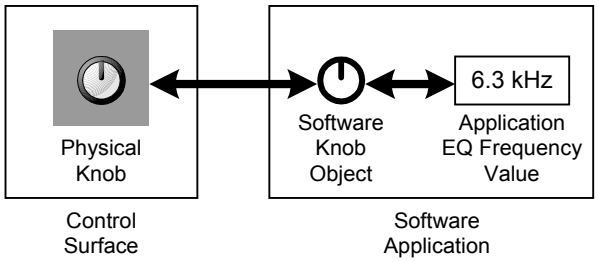

Figure 1: Software knob controlling EQ

Communication between the control surface and application is bi-directional. When the application wants to change a parameter, it calls a method on the EuCon software object. The physical surface control will change accordingly. In the other direction, when the user changes a control on the surface, EuCon generates a callback in the application, which then changes the application parameter.

For a large project, a DAW may model thousands of controls. This is far more than there are physical controls on a typical control surface, as depicted in Figure 2.

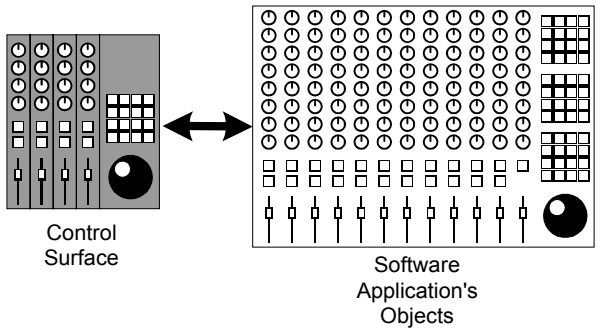

Figure 2: Fewer controls on surface than in application

The surface is able to find meaningful controls in the application and selectively map them to controls on the surface. Exactly how EuCon accomplishes this will be discussed in the remainder of this paper.

5. OBJECTS

An object-oriented interface for the protocol was chosen because hardware controls found on mixing consoles and audio processors fit neatly into the object paradigm. Channels, busses, faders, knobs, and meters are all easily comprehended as discrete, cohesive objects, making programming easier. Also, consoles and many DAWs have a modular hierarchical design that lends itself well to object modeling. For example, a console includes many replicated channels, each channel including a fader which includes a level control slider, mute switch, and so on. Using objects, the programmer models the entire portion of their application's user interface that is desired to be controlled by a hardware surface.

EuCon objects fit into a containment hierarchy. At the lowest level are *primitive* controls, which are contained by *controls*, which can optionally be contained in a *control array*. Controls and control arrays are contained by *processors.* Each of these levels, shown in Figure 3, will be explained in the following sections.

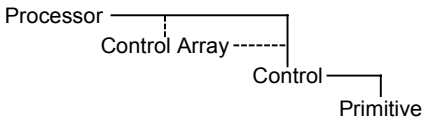

Figure 3: Object containment hierarchy

5.1. Primitives

The most basic software object in EuCon is a primitive control, which will be referred to hereafter as a *primitive*. It represents an actual physical control. EuCon defines twelve primitives which are shown graphically (for illustrative purposes) in Figure 4.

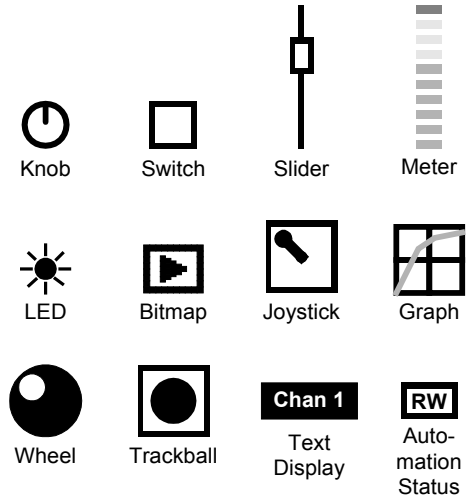

Figure 4: Primitives

These twelve primitives can be combined to create user interfaces that map to a wide variety of control surfaces. Some, such as the graph and bitmap, are unique to EuCon.

5.1.1. Primitive Values

The primitive maintains a value, which represents its current state. Supported value types are:

- 32-bit integer number
- 32-bit floating point number

- String

A slider primitive value, for example, might be represented by a 32-bit floating point number that ranges from a high value of +12.0 dB to a low value of –200.0 dB (off).

A text display primitive, on the other hand, is always represented by a string value. A channel name on a mixer strip is an example.

Physical control surface text displays come in a variety of text widths. Because of this, EuCon's string value type consists of three separate strings - a short string for display on four character physical displays; a medium length string for eight character displays; and a long string for longer length displays such as those found on active matrix screens. This allows the string to be properly displayed on a variety of control surface text displays.

There are two ways to represent a primitive's value. It can either be directly represented by an integer, float, or string value, or it can be indirectly represented by an index into a table. A table contains all possible values for the primitive. The table can contain integer or floating point values as well as string values. The primitive maintains a 16-bit current *index*, which is an offset into the table. Figure 5 shows an example of a table containing filter types, such as might be found on an Equalizer knob.

index	4char	8char	long
0	"lCut"	"Low Cut"	"Low Cut"
1	"loSh"	"Low Shlf"	"Low Shelf"
2	"Peak"	"Peak"	"Peak"
3	"hiSh"	"HighShlf"	"High Shelf"
4	"hCut"	"High Cut"	"High Cut"

Figure 5: Primitive value table

Using a table allows for non-linear value ranges, such as piecewise linear fader tables or logarithmic frequency tables.

Each primitive has methods to get and set its values, its current index, and its table values.

The primitive's 32-bit integer and floating point values provide much higher resolution than the seven bit values found in some MIDI protocols.

5.2. Controls

A *control* is an object that contains one or more primitives. A switch control, for example, contains both a primitive switch object and a primitive Light Emitting Diode (LED) object as depicted in Figure 6. The LED is typically used to show the state of the switch, but can be set independently if desired.

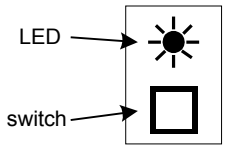

Figure 6: Switch control

Controls provide a way to aggregate the primitives, reducing complexity. A knob on a large format mixing console, for example, may contain more primitive controls than a single knob. It may also contain a text display to show the knob function, a touch switch to indicate that the user is touching the knob, and a switch to set the automation mode of the knob.

EuCon currently provides 15 predefined controls – switch, knob, fader, text display, automation status, bitmap, dual joystick, graph, knob cell, LED, meter, multimeter, slider, trackball, and wheel. Four often used controls are listed in Table 1, along with the primitives they contain.

Control	**Contained Primitives**
Text Display	*text display*
Switch	*switch*, *LED*
Knob	*knob*, label *text display*, touch *switch*, knob top *switch*, automation status *switch*, *automation status* display, function *LED*
Fader	*slider*, touch *switch*, backstop *switch*, select *switch*, select switch *LED*, automation select *switch*, *automation status* display, mute *switch*, mute *LED*, mute select *switch*, mute select switch *LED*

Table 1: Common controls

Not all of the primitives within a control need to be used, but they are available for user interfaces that need them. EuCon code is optimized to consume very little memory or computational resources for unused primitives.

5.2.1. Control Arrays

A control array is a control that contains other controls. There are two are predefined control arrays - a *switch array* and a *knob cell array*.

A switch array contains any number of switch controls. Switch arrays are often used to aggregate controls that can be presented as menus by the control surface. For example, a common use of switch arrays is to provide access to every command found in an application's GUI menus, such as File, Edit, View, etc. All of the File commands, such as New, Open, Close, etc., can be modeled as switches and put into a single switch array representing the File menu. The surface can take advantage of this and present a menu system to the user containing all of an application's controls.

A knob cell array contains *knob cell* controls, which in addition to the primitives in a knob control (see Table 1) have an upper and lower switch, as are sometimes found on large format mixing consoles. A knob cell is depicted in Figure 7.

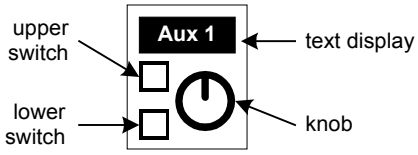

Figure 7: Knob cell control

An auxiliary send in a DAW can be modeled with a knob cell. The knob is used to adjust the send level, the lower switch for toggling the send on/off function, and the upper switch for toggling the pre/post function.

A knob cell array is used to model *knob sets,* which are a collection of similar parameters. Continuing with the auxiliary send example, all of a DAW channel's sends can be modeled as an array of knob cells. For example, if an application supports five auxiliary sends per channel, then a knob cell array can be used to model all of those sends, as depicted in Figure 8.

Figure 8: Knob cell array

In addition to auxiliary sends, knob sets can be used to model signal processing algorithm controls, such as for panning, equalization, dynamics, and plug-ins. Knob sets can also be used to model channel input and output routing controls. Knob sets are displayed on control surfaces on an array of physical knob cells. If the size of the knob set exceeds the number of physical knob cells, then the user can page through the knob set to reach any given knob cell.

Knob cells in a knob cell array can in turn contain other knob cell arrays, creating arbitrary levels of hierarchy. This feature is useful when modeling complicated plug-ins that have dozens of parameters. Similar parameters can be grouped together in separate child knob sets. To access a child knob set from a control surface, the user presses the physical knob, which contains a switch, called the knob top switch (see Table 1).

5.3. Processors and Surfaces

EuCon uses the term *processor* to refer to the application, and *surface* to refer to the control surface. The term *processor* was chosen because nearly every application - DAW, microphone preamplifier controller, video editor, or signal processing effect - is processing data in some way.

A *processor* object contains one or more controls. There can be many processor objects in an application. Processor objects have a type. Examples of different types of processor objects include a channel strip, which typically contains a fader, many knob sets, and other controls; and a transport, which contains play, stop, and other switches. The processor object is subclassed to create various types of application-specific objects.

Figure 9 shows a diagram of a simple channel strip that contains a fader control, a switch control, a text display control, and a knob cell control.

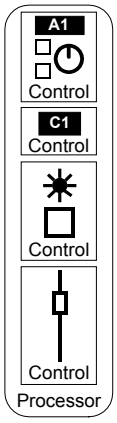

Figure 9: A channel strip processor

The control surface also has a *surface* object analogous to the processor object. The control surface creates surface objects that contain controls and primitives that model its physical user interface.

5.3.1. Callbacks

Processor objects have a callback method that is invoked whenever a contained primitive object changes state in response to its physical surface counterpart being moved. The programmer overrides the callback method to change application parameters accordingly. Arguments are passed into the callback method indicating the primitive that changed, its containing control, and its new value. The application uses these arguments to update the application parameter that corresponds to the moved physical control. The application also has the ability to validate the incoming value, clipping it if necessary. EuCon then returns the clipped value back to the surface.

5.4. Nodes

Every application has a single *node* object. All processor objects *register* with the node. This way, the node is aware of all processors and hence their contained controls and primitives. Every control surface is also represented by a node object. Node objects are used to identify surfaces and applications when connecting them together.

5.5. Attributes

Node, processor, surface, control, and primitive objects can all be tagged with one or more *attributes*. An attribute is a key-value pair, where the key can be either an integer or a string, and the value can also be an integer or string. There are several dozen predefined attribute keys. A common use for attributes is to apply a user-visible string name to a control, such as a switch. This string name is used on control surfaces with dynamically changeable switch labels to show the name of the switch.

6. CONNECTING SURFACES TO WORKSTATIONS

The preceding section described how to use software objects to model an application's user interface. This and the following sections describe how surfaces find applications and then map surface controls to the application's modeled objects in an intelligent manner.

One of EuCon's design goals is to allow many surfaces to control a single application. This allows a central surface and several satellite fader units to control a DAW. Additionally, one surface should be able to control many applications. That way, a user can simultaneously control a video application and an audio application from a single surface or a set of cooperating surfaces. This feature is important as multimedia production becomes more pervasive.

EuCon uses the term *workstation* to refer to a computer that runs an application. A workstation doesn't have to be a personal computer; it could also refer to an embedded computer running in a processing device. Control surfaces typically have an embedded computer running inside. All control surfaces and workstations are connected to a common TCP/IP network. There is one node per control surface, and one node per application. There may be several applications, and hence nodes, on a single workstation. This topology is shown in Figure 10.

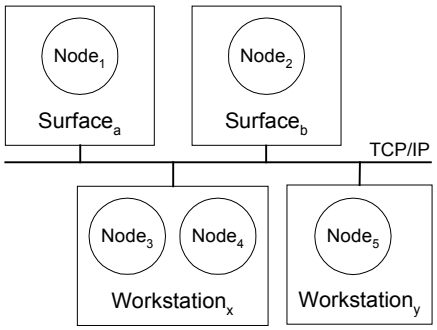

Figure 10: Network topology

All nodes, both in the surface and workstation, register with a distributed database called EuCon Discovery that runs in each surface and workstation. Using TCP/IP multicast protocols, EuCon Discovery maintains a replicated database of all nodes on the network. When a user requests that a surface control a node on a workstation, the surface looks up the node in EuCon Discovery's database and retrieves the workstation's IP address, enabling it to connect.

EuCon is based upon a distributed object system that allows the application's objects to have mirrored counterparts in one or more surfaces. The surface can make remote procedure calls on the application's objects to control them and visa versa.

7. ASSIGNING CONTROLS

Once connected via TCP/IP, a surface proceeds to map its controls to the application's controls. This process is called *assignment.*

7.1. Layout Rules

Each control object in an application has a predefined unique identifier assigned to it by the programmer, called a *layout name.* For example, there is a layout name for a play switch. It uniquely identifies the switch used to start playback in the application. The layout name allows the surface to map its own controls to the corresponding control in the application.

EuCon defines several categories for layout names called *layout rules*. Some of the more common layout rules and names are listed in Table 2.

Rule	Names
Channel	Fader, On, Solo, Name, Select, Record Arm, Meter, EQ Knob Set, EQ Graph, Dyn Knob Set, Dyn Graph, Pan Knob Set, Pan Graph, Inserts Knob Set, Input Routing Knob Set, Output Routing Knob Set
Transport	Play, Stop, Record, Fast Forward, Rewind, Pause, Cycle, Primary Time Display, Left Locator, Right Locator
Monitor	Control Room Volume, Mute, Dim, Control Room Source, Main Spkrs, Alt1 Spkrs, Alt2 Spkrs, Monitor A Volume, Monitor A Source, Monitor B Volume, Monitor B Source, Talkback
System	Clear Mute, Clear Solo, Automation Mode, Open Plugin Windows, Close Plugin Windows

Table 2: Common layout rules and names

To assign its controls to an application, the surface iterates through all of the application's controls using objects called *browsers*. Starting with the application's node, the surface can browse down through every processor to its controls and examine their layout names. As it finds controls it wants to assign to, the surface creates a mirror object of the application control using the distributed object system. The mirror allows the surface to make remote procedure calls on the application object, enabling bi-directional communication over the TCP/IP network. To complete the assignment, the surface maps its own physical control to the mirror of the application's control.

An application's object model can dynamically change in response to user actions. For example, when a user adds a plug-in to a channel strip in a DAW, a new knob cell array is added. Removing a plug-in deletes a knob cell array. Using a callback, EuCon notifies the surface when additions, deletions, or modifications occur. The surface can then assign to newly added controls, unassign from deleted controls, or update modified controls.

7.2. Well-Known Processors

Each processor object is tagged with an integer value attribute called a *processor type*. When a surface assigns to a processor, it looks for these types and maps certain controls accordingly. Well-known predefined processor types also aid ease of programming by creating a level of modularity – like controls are contained in each processor type. Common processor types are described in the sections below.

7.2.1. Channel Strip Processor

A channel strip processor type tells the surface that the processor contains controls that will be mapped to a single channel strip. There is one channel strip processor instance for each channel (sometimes called a track) in the application. All controls in a channel strip use the channel layout rule (see Table 2). A channel strip processor has an integer value attribute set on it that specifies the order to present channel strips on the surface – lowest number in the leftmost strip to highest number in the rightmost strip. There are cases where an application has more channel strips than exist on a control surface. The surface can implement a policy to handle this, such as scrolling through the application's channel strips. Another policy is to allow the user to directly assign an application's channel strip to a user-chosen strip on the surface.

7.2.2. Command Processor

The command processor holds an application-defined set of switches that can be invoked using control surface *soft keys*. A soft key is a switch that has a display on it so it can be relabeled.

Typically an application will model all application menus (such as File, Edit, etc.) as switch arrays and model menu items (such as New, Open, Close, etc.) as switches (see section 5.2.1). Some DAW applications create more than 2,000 switches in the command processor.

The control surface allows the user to place switches on soft keys using a configuration utility, which displays three levels of hierarchy:

1. processors, which contain
2. switch arrays, which in turn contain
3. switches

The processors, switch arrays, and switches are given user visible names using a string value attribute. By convention, the command processor is given the name "Key Commands". Figure 11 shows the configuration utility of an existing EuCon control surface.

Figure 11: Soft key configuration utility

7.2.3. Transport Processor

The transport processor contains switches for Play, Stop, Record, etc. (see Table 2). It also contains text displays for time code and locator points. It contains a switch array, named "Transport", so the transport keys can be placed on control surface soft keys as well as on fixed physical keys. This switch array can contain any number of application specific switches. Additional switch arrays can be added, extending the surface functionality even further.

There is only one transport processor per application. Multiple transports, i.e. control of multiple machines, can be modeled with a switch array per transport.

7.2.4. Edit Controller Processor

An edit controller processor contains a jog wheel and jog/shuttle mode switches. Applications are encouraged to support as many functional modes for the jog wheel as possible, including horizontal and vertical zooming, waveform zooming, trimming of clip head and tail,

cross fade length, and clip gain. Accordingly, the edit controller processor contains a switch array containing switches to set these modes. By convention, the edit controller processor is named "Edit Control". Some control surfaces support two physical edit controllers, a left and a right. The application can choose to model both controllers, taking full advantage of these surfaces. In such a case, one editor controller processor is named "Left Edit", while the other is "Right Edit".

7.2.5. Monitor Processor

The monitor processor contains controls found on a large format mixing console's monitor section, such as a knob for the main control room volume, main volume dim and cut switches, speaker selector switches, and switches to select the audio sources for the control room (see Table 2). Like any other processor, it can contain any number of additional switch arrays containing application-specific functions. There is one monitor processor per application.

7.2.6. Project Processor

The project processor contains controls specific to an application's open document, sometimes called a *project*. It contains a switch array containing time line marker switches. Pressing one of these switches moves the transport's "play head" to the marker's position in time.

7.2.7. System Processor

The system processor contains global controls such as the clear mute, clear solo, and automation mode switches (see Table 2). It also contains switches that reflect preferences set by the user from the control surface, such as whether to open plug-in windows or not when editing a plug-in using a knob set. There is one system processor per application.

8. MULTIPLE APPLICATIONS

Audio and video content creators often use several applications at once, such as a video player and an audio editor. EuCon was designed to allow a control surface to simultaneously control an unlimited number of applications. There are several ways this is done - application switching, control locking, and workstation switching - each explained below.

8.1. Application Switching

By default, the control surface controls the *in focus,* or topmost, application on a workstation. The in focus application is the one receiving mouse and keyboard commands.

A EuCon-supplied program running on each workstation uses operating system routines to determine which application is in focus. When focus changes, this program notifies the EuCon runtime, so the control surface can reassign to the newly focused application.

8.2. Control Locking

Portions of a control surface can be *locked* to one application while other portions of the same surface are left to freely control the topmost application. One use is to lock the transport section of a control surface to a master video editor while keeping the rest of the surface assigned to an audio editor. Another example is using one DAW for playback and another for mixing. Some channel strips can be locked to the first DAW while the rest are used to control the second.

To accomplish this, the control surface maintains mirrored objects for both applications. It maps one set of physical controls to one application and another set to the other application.

8.3. Workstation Switching

A control surface can quickly switch between multiple workstations. Some EuCon control surfaces have dedicated keys for this task. When one of these keys is pressed, the surface will reassign to the new workstation. Sections that are locked will remain locked to the original workstation, allowing locking to span multiple workstations.

8.4. Unified User Interface

Each application has its own GUI, which typically differs from application to application. By following the conventions encouraged by layout rules and well-known processors, a single control surface is able to provide a unified user interface to each application.

This results in more efficient workflow compared to using the different GUIs without a control surface.

9. CONCLUSION

Using EuCon, software applications can support large, sophisticated, and highly evolved user interfaces on control surfaces similar to those found on large format mixing consoles. It has ample bandwidth, high control resolution, and supports a wide variety of user interface controls. Its object-oriented design makes it easy for developers to adopt. It supports switching between and controlling multiple applications simultaneously. The protocol has been deployed since October 2005 and can currently control a wide variety of commercial applications.

10. ACKNOWLEDGEMENTS

The authors would like to acknowledge Hans-Joerg Ziegler for his contributions to EuCon.

11. REFERENCES

[1] Sanders, M. S., and McCormick, E. J., "Human Factors in Engineering and Design", McGraw-Hill, Inc., New York, pp. 47 - 90, (1993)

[2] Hawkins, E., "Stop Mousing Around!", Mix Online, (May 1, 2003) (http://mixonline.com/products/buyersguides/audio_stop_mousing_around/index.html)

[3] "Control Surface Reviews and Tutorials", Electronic Musician Web Site, (June 2006) (http://emusician.com/controlsurfaces/)

[4] Wherry, M., "Mackie Control DAW Control Surface", Sound On Sound., (December, 2003) (http://www.soundonsound.com/sos/dec03/articles/mackiecontrol.htm)

[5] Wright, M., Freed, A., Momeni, A., "OpenSound Control: State of the Art 2003", Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME-03), Montreal, Canada

[6] "Jazz Mutant Lemur" , Keyboard Magazine Online Edition (February 2005) (http://www.keyboardmag.com/story.asp?storycode=184)

[7] "John Ross Creates Mixing Environment with Gefen Switchers and Euphonix System", Mix Online, (May 24, 2006) (http://mixonline.com/news/headline/johnross-euphonix-gefen-052406)

[8] Grossberg, M., Wiesen, R. A., and Yntema, D. B., "An experiment on problem solving with delayed computer responses", IEEE Transactions on Systems, Man, and Cybernetics, vol. 6, no. 3, pp. 219 –222, (March 1976)

[9] Becka, K., "Technology Spotlight: Digidesign ICON", Mix Guides, (April 1, 2004) (http://mixguides.com/consoles/product_features/technology-digidesign-icon-0404/)

[10] Postel, J. (ed), "TRANSMISSION CONTROL PROTOCOL", RFC 793, Defense Advanced Research Projects Agency, (September 1981)

[11] Metcalfe, R. M. and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks", ACM Communications, vol. 19, no. 5, pp. 395-404, (July 1976)

[12] "1394-1995 IEEE Standard for a High Peformance Serial Bus – Firewire" (http://standards.ieee.org/)

[13] "Universal Serial Bus Specification, Revision 2.0" (April 27, 2000) (http://www.usb.org/developers/docs/)

[14] Stroustrup, B., "The C++ Programming Language (3rd Edition)", Addison-Wesley Professional, 3rd edition (June 20, 1997).

[15] Booch, G., "Object-Oriented Analysis and Design with Applications (3rd Edition)", Addison-Wesley Professional; 3rd edition (June 4, 2004), pp. 35-37.