Bias, Reuse and Interface Representation.

Robert Shelton
Department of Computer Science
University of Melbourne
Victoria, 3010
Australia

rshelton@cs.mu.oz.au

Abstract

This paper introduces the Environment for a Semiotic Control Interface (ESCI). ESCI inherits the motivation of previous systems such as W and the Open Sound Control (OSC), while extending their functionality to enable a broader range of type definitions and peer-to-peer collaboration. ESCI allows the passive structures of applications to be explored in similar fashion to the OSC namespace. ESCI adopts a semiotic approach in which the cooperative interaction of OSC is extended through the possibility of competitive and symbiotic interaction.

Introduction

The chief task of a computer music user-interface is to assist in the difficult translation of musical, intuitive and informal ways of thinking to a formal representation (Oppenheim 1991b). Currently there is no single target representation. Instead there exist many computer music application and interface designs each presenting their own musical ideas through the software (Puckette 2002). As a result, every system or interface tends to enforce a method of description which then constrains or biases the composer's view (Roads 1996).

This situation is constantly improving as each evolution of computer music software provides a more powerful language, a better user interface or more robust facilities. Yet as the interface becomes more specialized, its lifetime is reduced by increased complexity, cost of maintenance or an intrinsically tight binding of the code to hardware (Bernardini 2002).

The poor rate of software survival has led to a situation in which simpler, modular and customisable components are used in conjunction to form a more complex system (Polfreman 1995). Examples include single applications like CSound making use of the unit-generator abstraction, or multiple applications cooperating through established interfaces such as the Musical Instrument Digital Interface (MIDI) or OSC.

Collaboration between components has been described as a software ecology. Furthermore, it has been suggested that in order to move collaboration in the field of computer music to a higher level, a system is needed that facilitates the development of software ecologies at many levels (Zicarelli 2002).

This paper introduces the Environment for a Semiotic Control Interface (ESCI) aimed at exploring the possibility of using a semiotic control approach for computer music. In order to observe application behaviour, ESCI promotes shared access to passive and internal data structures. Although these passive structures have a formal meaning with respect to the original application design, there is no reason that they should not be reused in novel or creative ways. This point of view corresponds to the idea of a performer interpreting the formal representation of traditional music.

> "Insofar as every situation in a musical piece may (or may not) announce a foreseeable but unpredicted musical solution, music offers another example of a semiotic system in which each situation could be differently interpreted."
> (Eco 1976)

The next section introduces the possibility of a range of low to high-level computer music representations. Computer music control is then introduced as a method for interacting or collaborating at a particular level of representation. This results in the need for interface descriptions of either the passive structures or the active methods. A semiotic approach is then considered along with the new modes of interaction it provides. Finally an overview of ESCI is presented with respect to reliability and architectural considerations.

Computer Music Representation

When introducing a panel discussion at the 2002 Dartmouth Symposium on the Future of Computer Music Software, Eric Lyon (2002) suggested that perhaps the history of computer music languages and systems is more or less about survival paradigms. This focus on software survival reflects the amount of time required for computer music systems to mature and become accepted by the wider community.

Lyon advocated that CSound (Music-N), Max/MSP, and SuperCollider can be considered to represent the three surviving paradigms of computer music. Although very different systems, it is reasonable to suggest that all three are remarkably similar. They all convert a visual or textual language into a network of connected unit generators. Within CSound

this is the orchestra file, MAX/MSP uses such a network as a visual interface, and the SuperCollider Server abstracts this aspect out of the SuperCollider client.

It is reasonable to suggest that for any sample produced by the synthesis algorithm, a finite function must have been applied to generate it. The network of unit generators provides the description for such a function. However, this abstraction only applies for a single sample and thus only a single instance of time.

Temporal control of the unit generator network can be achieved using the score in CSound or an object-oriented language similar to Smalltalk in SuperCollider. Max uses what has been called a resource model (Dannenberg 1991) which is different again.

All three paradigms control a network of unit generators through their own music representation. As a result, all three are forced to provide solutions to the associated problems such as style of description, representation of time, hierarchy of representation and the provision for multiple views of representation (Dannenberg 1989).

The survival of these systems validates the representation and interface adopted. The differences between them suggest that the possibility of a single representation is very unlikely. As a result, these systems remain isolated and provide little assistance for new developments outside of their existing frameworks.

An interface representation which could be used between systems is possible, but likely to provide only the better understood subset of functionality (Openheim 1991a) in which novel composition is outside the scope of the formalism (Helmuth 1996). An alternative is to construct a single monolithic computer music application. However, this can limit the composer when attempting innovative or unorthodox works (Roads 1989), and hamper development given that there seems no practical method to include every individual idea for the reliable benefit of all (Vercoe 1996).

Within this section a single low-level representation has been suggested as well as the unlikely creation of a single high-level representation. An environment aiming to assist development and maintenance for new representations or interfaces will need to find a niche somewhere between these two extremes. The subsequent section argues that computer music control provides a good target level.

Computer Music Control

The variety of modes in which computer music applications are used leads to a number of possible definitions of control. For the purpose of this paper, control is defined as the correlation between the physical parameters or interface of the system and the perceptive concepts or intention of the composer. Given this definition, control has been suggested as a major problem for computer music applications (Oppenheim 1986).

A substantial part of this problem relates to the volume of data required to realize music of any subtlety (Loy 1989). Furthermore, the introduction of real-time control introduces a new perspective on composition (Lefford 1999) encouraging a preference for lower latency, higher precision and higher data rates (Wessel 2002). With this new perspective comes the need to reconsider the traditional split of representation into active procedures and the passive data structures that support it (Roads 1989).

Another aspect of real-time behaviour is that control is not always a human-machine interface. Control can also be exerted between machines using protocols such as MIDI or OSC. This introduces the possibility of multiple machines which in turn allows multiple users. As a result, there is no longer a single interface, but instead a network of, or perhaps a distributed system for control. This promotes new possibilities for collaboration between composers, performers, developers and machines.

A group and system known as The Hub (Collins 2003) provides a good example of this. Making little distinction between the method used for composition or performance, The Hub adopts a framework in which a single machine is dedicated to passing messages between multiple machines. This message passing interface supports the individual contribution to the composition or performance by allowing each member a better understanding of the application's distributed state.

The Hub represents a custom solution to a specific control problem. As a result, its ability to support a wider range of control problems is limited. MIDI is in a similar situation (Chabot 1985). In order to generalise the framework a method is required to describe the interfaces provided by different components of the computer music system.

Interface Description

The traditional split of computer music representation into active procedures and passive data structures is undoubtedly a result of software-engineering methodology – in particular object-oriented design. In this section it is argued that traditional active interfaces, such as the Common Object Request Broker Architecture (CORBA), do not necessarily provide the best interface description for computer music applications. Instead passive interfaces such as W or the Open Sound Control (OSC) are considered better alternatives.

Traditional Active Interfaces

In line with the aims of software-engineering, the object-oriented paradigm is designed to simplify development, ease maintenance and hopefully improve the possibility of component reuse. One method of achieving this is through the encapsulation of passive data behind the object's active interface. These objects can be organised into a hierarchy which should reflect the structure of the applications.

Using object-oriented techniques for computer music representation has been a prominent approach when trying to providing multiple views of a single component (Roeder 1989), manipulation of application components (Holland 1999), and representation portability between different applications (Dyer 1989). However, the success of such developments is not clear. In fact, for a long time it has been suggested that a "toolbox" would be more useful than a closed system (Assayag 1986).

Some more recent developments in computer music have adopted CORBA as such a toolbox (Behles 1996; Assayag 1996). CORBA allows an application developer to define an active (method based) interface within an object-oriented environment. Access to passive data can then be provided through this active interface. The primary feature of CORBA is its programming language and operating system independence. This allows it to be incorporated into almost any style of application. Also the interface defined is then available across the network which ties in well with the idea that control is a problem for distributed systems.

There are also a number of negative aspects relating to the use of CORBA. As a result of its generality and maturity, CORBA is a large and intricate system. Learning to use it well is a difficult task and application development becomes more complex as the original design must take into account the requirements of CORBA.

Since many original computer music applications are exploratory and attempt to prove a concept, representation or interface, it becomes unlikely that the time investment required to adopt a CORBA framework will be considered reasonable. Modifying proven applications to use CORBA would seem just as unlikely. A simpler interface will have a smaller influence on the design of a new application; furthermore, it will be easier to incorporate into existing and proven applications. As a result, simplicity becomes a key factor in determining the uptake of such interfaces.

Two Passive Interfaces

In opposition to the complexity of an object-oriented paradigm, there exist two systems which demonstrate an alternative: W (Dannenberg 1995) and OSC (Wright 1997). Both can be considered an interface to the passive components of computer music, because both are based on the idea of assigning and retrieving values with basic types using asynchronous messages. Therefore, it is more a process of retrieving or setting object values, rather than calling object methods. Of course the act of setting values is also an event which can trigger a return value to be sent, so in this respect methods can be simulated within the original framework.

OSC provides less overall functionality than W and also makes few assumptions regarding flow of program control. As a result, it is much easier to integrate OSC into existing applications. This has included CSound, Max/MSP, SuperCollider and many more (Wright 2003).

Objects in an OSC environment are dynamically organised into a hierarchy (namespace) which reflect the features of the application (Wright 1998). A foreign namespace can be explored to reveal the objects, and their respective types, within the hierarchy. As a result, OSC enables software to be self-documenting and transparent in its functionality (Wright 2003).

If control is the correlation between physical parameters of the system and the perceptive concepts or intention of the user, then this ability to explore the system's functionality becomes valuable. Exposing the system to the user and thus providing a path for the "user to become the developer" (Nieberle 1988) is of significance in an environment where the original developer can not provide the functionality or representation required by all computer music users.

This mixing of roles – where the user is the composer, performer and developer – creates a new situation for user-interface design. In a creative situation where there is no limited set of tools, and also no single way for those tools to be applied, traditional user interface design becomes less appealing. If functionality of the system is explorable and modifiable, then the user gains a enhanced capacity to create a personalized environment. Respective interface features can be arranged into a pattern which more adequately reflects the user's own perspective (Todoroff 1997).

In this section active and passive interfaces were introduced. Although it was primarily described from a software development perspective, many of the issues raised influence user-interface design and can promote new opportunities for user control. The next section explores the idea that these new opportunities move the previously symbolic manipulation of computer music applications, to something more semiotic.

Semiotic Control

The passive interfaces described are only of use to the software developer. In order to control this aspect, a human-machine interface is required. From a semiotic standpoint, the connection between these two interfaces requires a "code" (Eco 1976) that apportions the control system to the elements of the musical system (Roeder 1989).

OSC provides the transport for this code. When applications such as CSound, Max/MSP and SuperCollider incorporate the OSC system, collaboration over this transport becomes possible. However, from a semiotic perspective the current paradigm would seem to only support cooperative behaviour but not two other suggested modes of interaction: competitive and symbiotic (Iazzetta 1996).

In cooperative interaction all components of the system share the same goals. This would seem to be the only form of interaction currently available since

all collaboration between components is arranged at a global level by a standardised control mechanism. The control provided by OSC represents an abstraction from the complete system behaviour to a simplified subset of functionality. Furthermore, manipulation of that functionality is generally expressed in the terms of the existing computer music environments.

Competitive interaction suggests a situation where the success of one component implies the failure of others. Symbiotic interaction suggests the existence of different goals but the ability to share the same context or environment. Although probably less useful than cooperative interaction, these two can be interpreted to represent unexpected reuse of the system or user reshaping of the environment. Both of these interactions can allow the user access to normally hidden parts of the system, the ability to separate components into smaller parts and reinterpret the passive representation as desired.

By disrupting the traditional user-interface paradigm, an interface based on a historically determined representation of the problem can encourage creative problem-solving (Traux 1986). Therefore, creative control over the environment promotes the user's ability to imagine and implement musical processes that otherwise may not be feasible (Hamman 1999).

An abstraction from the functionality can mostly provide an expected and cooperative interaction. In order to provide more competitive or symbiotic interactions – and in turn support a semiotic environment – the ideas of W and OSC need to be extended. In the next section ESCI's approach to such an extension is introduced.

Approaching a Semiotic Environment

This section introduces ESCI (Environment for a Semiotic Control Interface). ESCI represent one approach to the difficulties involved in providing the semiotic interactions described in the previous section. It does not aim to develop a user interface model, but instead exists as a code transport between distributed computer music applications and user-interfaces. Neither the design or implementation of ESCI is complete; nevertheless, this description is considered essential in presenting an overall view of the problem and allowing the reader to determine whether the current approach is satisfactory. The description of ESCI is broken into two sections – an overview of the reliability concerns followed by a description of the basic system architecture.

ESCI Reliability

ESCI aims to be a straightforward framework for developers to incorporate in to existing applications. The distinguishing attribute is that ESCI promotes the sharing of passive application structures. This is in opposition to W and OSC which share an abstraction of the complete application functionality through a passive interface. As a result, the user (and other applications) are no longer exploring an abstraction based on the expected use, but instead have access to explore the internals of the computer music application.

This intention disregards a substantial amount of software-engineering dictum, especially with regard to reliability of such an open (and thus chaotic) system. However, the cooperative interface within ESCI will be as safe as the equivalent interface found within OSC. It is competitive and symbiotic interactions which are more unpredictable and better suited to the more experienced users. In fact, they should probably be hidden from the novice user interface (Cascone 2000).

A semiotic interface provides a path for the user to become the developer. ESCI aims to enable users to extend the functionality of the application without access, or modification, to the source code. As previously mentioned maintaining a reliable base application is difficult in the face of user contributions (Vercoe 1996) and forking application development for specialist purposes is not a sustainable solution. It is expected that any other method trying to provide similar functionality will face similar concerns.

ESCI Architecture

In this section a brief overview of the ESCI architecture will be introduced. A fundamental aspect of ESCI is the simplicity with which it can be incorporated into existing application. As a result, little or no effect on the flow of program control can be assumed. The long term ambition would be to automate the procedure of incorporating ESCI into an existing application. It is also expected that an environment such as ESCI has the opportunity to influence software-engineering techniques to be more accepting of semiotic interaction (especially with respect to remote access). In parallel with these aims, the design of ESCI is operating-system, hardware and language independent.

The description of ESCI is broken up into three parts: sharing passive representation, providing type information, and remote access.

Sharing Passive Representation

Application source code is a static description for the temporal behaviour of a program. As a result, the passive structures of the program provide the best opportunity to understand the dynamic behaviour of the application at any point in time.

OSC provides access to these structures via message passing; however, this interrupts the flow of program control as messages must be received and processed. An established alternative to a message passing system is a shared memory system.

ESCI provides a scalable lock-free memory allocation algorithm to make interesting passive structures visible. This policy is similar to W (Dannenberg 1995) and thus ESCI is also forced to cope with the added

address space complexity. Existing applications are not effected by this, but exploration of another program's address space needs to consider such issues if pointer types are shared (a capability not provided in OSC).

Providing Type Information

Although in some cases the type of a data could possibly be identified through user observation and exploration, ESCI provides a dynamic type system. Applications can indicate the type of a shared passive structure. Type information defines not only the form of the structure but also an expectation of how it should be used. This improves the reliability and efficiency of working with passive representations.

The basic form of ESCI's dynamic type system is inspired by established network type definitions such as the External Data Representation (XDR) or the Abstract Syntax Notation One's (ASN.1) Basic Encoding Rules (BER). Passive structures can be defined in terms of base types and other passive type definitions. This produces a hierarchy of types equivalent to those found in programming languages. In fact, most if not all of the type information can be gleaned from the application source code.

A more complete type system is considered a chief contribution of ESCI. It allows for the possibility of passive structures such as text-buffers, queues and trees to be shared. The primary user-interface for many applications is a textual representation with an associated syntactic interpretation (parse tree). Providing a method of sharing such representation would seem to be a required feature of any new environment.

Remote Access

Without modifying the flow of program control, or even the original source code, the task of making a previously local application distributed is an interesting research problem. ESCI approaches the problem from a distributed shared memory view.

ESCI runs a network process on each machine which services asynchronous messages requesting local passive structures to be queried or assigned. This service is similar to that provided by OSC but not identical. For example, because OSC handles the messages within the application, it is possible for multiple objects to be set atomically. Within shared memory only a single memory word (such as a single integer or float) can generally be modified atomically.

Furthermore, because the messages are handled within the application, modification of the passive representation is easy to detect. Otherwise there is a need to mark the passive structure to indicate that a change has occurred. Again this is an established problem within distributed shared memory, as changes to shared objects need to be communicated to other machines holding a copy of that object.

Marking the passive representation (or generating a message) to indicate a modification has occurred requires the flow of program control to be modified. OSC demonstrates that as long as the modification is not substantial then applications are willing to incorporate such changes.

OSC is also in a simpler position because messages represent one-to-one communication. Shared memory enables the possibility for multiple applications to interact simultaneously providing a more developed environment for computer supported cooperative work. The challenge of providing an adequate marking technique is a cornerstone of the ESCI architecture.

Conclusion

The prospect of a single formalism for computer music representation is very unlikely to ever exist. However, reuse of computer music components requires some aspects of representation to be agreed upon. The possibility of a single interface which provides optimal control for computer music applications is just as unlikely, since every interface can bias the composer's, performer's and developer's view of the system. Within this context, issues relating to bias, reuse and interface representation have been explored.

In response, the Environment for a Semiotic Control Interface (ESCI) has been introduced. ESCI inherits the motivation of previous systems such as W and Open Sound Control (OSC), and attempts to extend the functionality provided to computer music application developers and users. ESCI allows the passive structures of applications to be explored in similar fashion to the OSC namespace. A semiotic approach is adopted in which the cooperative interaction of OSC is extended by the possibility of competitive and symbiotic interaction. The ESCI architecture contributes truly passive representations, more substantial type information and better sharing between local and remote applications.

ESCI is an evolving environment for distributed user and component collaboration. As documented, a number of research questions remain for this project, and any other project attempting similar functionality. Nevertheless, the aim of breaking apart applications and making their representation visible to other composers, performers and developers is considered essential for facilitating the development of software ecologies at many levels.

Acknowledgment

The author would like to acknowledge the reviewer for the insightful comments and suggestions.

References

Assayag, G. & Agon, C. 1996. "OpenMusic Architecture", *Proceedings of the International Computer Music Conference*, pages 339-340, August.

Assayag, G. & Timis, D. 1986. "A ToolBox for Music Notation", *Proceedings of the International Computer Music Conference*, pages 173-178, October.

Behles, G. & Lundén, P. 1996. "A Distributed, Object-Oriented Frame-work for Sound Synthesis." *Proceedings International Computer Music Conference*, pages 253-256, August.

Bernardini, N. & Rocchesso, D. 2002. "Making Sounds with Numbers: A Tutorial on Music Software Dedicated to Digital Audio", *Journal of New Music*, 31(2):141-151.

Cascone, K. 2000. "The Aesthetics of Failure: "Post-Digital" Tendencies in Contemporary Computer Music", *Computer Music Journal*, 24(4):12-18, Winter.

Chabot, X. 1985. "User Software for Realtime Input by a Musical Instrument", *Proceedings of the International Computer Music Conference*, pages 19-23.

Dannenberg, R. 1989. "Music Representation Issues: A Position Paper", *Proceedings of the International Computer Music Conference*, pages 73-75, August.

Dannenberg, R. & Rubine, D. 1995. "Toward Modular, Portable, Real-Time Software", *Proceedings of the International Computer Music Conference*, pages 65-72, September.

Dyer, L. 1989. "Position Paper for Music Representation Panel", *Proceedings of the International Computer Music Conference*, pages 98-100, August.

Eco, U. 1976, "*A Theory of Semiotics*", Bompiani.

Hamman, M. 1999. "From Symbol to Semiotic: Representation, Signification, and the Compositions of Musical Interactions", *Journal of New Music*, 28(2):90-104.

Helmuth, M. 1996. "Multidimensional Representation of Electroacoustic Music", *Journal of New Music*, 25(1):77-103.

Holland, S. & Oppenheim, D. 1999. "Direct Combination", *Proceedingsof the ACM SIGCHI Human Factors in Computing Systems Conference*, pages 262-269, May.

Iazzetta, F. 1996. "Formalization of Computer Music Interaction througha Semiotic Approach", *Journal of New Music*, 25(3):212-230.

Lefford, N. 1999. "An Interview with Barry Vercoe", *Computer Music Journal*, 23(4):9-17, Winter.

Loy, G. 1989. "Composing with Computers - a Survey of Some Compositional Formalisms and Music Programming Languages", in Mathews, M. & Pierce, J. editors, "*Current Directions in Computer Music Research*", chapter 21, pages 291-396. MIT Press.

Lyon, E. 2002. "Dartmouth Symposium on the Future of Computer Music Software:A Panel Discussion", *Computer Music Journal*, 26(4):13-30.

Wright, A. & Momeni, A. 2003. "OpenSound Control: Stateof the Art 2003", *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 153-159.

Rohrhuber, J. Collins, N. McLean, A. & Ward, A. 2003. "Live Coding in Laptop Performance", *Organised Sound*, 8:321-330.

Nieberle, R. Rothkamm, F. Verwiebe, M. Modler, P. Koschorreck, S. & Kosensky, L. 1988. "The CAMP system: An Approach for Integration of Realtime, Distributed and Interactive Features in a Multi-paradigm Environment", *Proceedings of the International Computer Music Conference*, pages 250-257, September.

Oppenheim, D. 1986. "The Need for Essential Improvements in the Machine-Composer Interface used for the Compositions of Electroacoustic Computer Music", *Proceedings of the International Computer Music Conference*, pages 443-445, October.

Oppenheim, D. 1991a. "Shadow: An Object-Oriented Performance System for the DMIX Environment", *Proceedings of the International Computer Music Conference*, pages 281-284.

Oppenheim, D. 1991b. "Towards a Better Software-Design for Supporting Creative Musical Activity", *Proceedings of the International Computer Music Conference*, pages 380-387.

Polfreman, R. & Sapsford-Francis, J. 1995. "A Human Factors Approach to Computer Music System User-Interface Design", *Proceedings of the International Computer Music Conference*, pages 381-398, September.

Puckette, M. 2002. "Max at Seventeen", *Computer Music Journal*, 26(4):31-43,Winter.

Roads, C. 1989. "Active Music Representation", *Proceedings of the International Computer Music Conference*, pages 257-259, August.

Roads, C. 1996. "*The Computer Music Tutorial*", MIT Press.

Roeder, J. & Hamel, K. 1989. "A General-Purpose Object-Oriented System for Musical Graphics", *Proceedings of the International Computer Music Conference*, pages 260-263, August.

Neuendorffer, R. Dannenberg, R. & Rubine, D. 1991. "The Resource-Instance Model of Music Representation", *Proceedings of the International Computer Music Conference*, pages 428-432.

Todoroff, T. Traube, C. & Ledent, J. 1997. "NeXTSTEP Graphical Interface to Control Sound Processing and Spatialization Instruments", *Proceedings of the International Computer Music Conference*, pages 325-328, October.

Truax, B. 1986. "Computer Music Language Design and the Composing Process", in Emmerson, S. editor, "*The Language of Electroacoustic Music*", Chapter 8, pages 155-173. MIT Press.

Vercoe, B. 1996. "Extended Csound", *Proceedings of the International Computer Music Conference*, pages 141-142, August.

Wessel, D. & Wright, M. 2002. "Problems And Prospects For Intimate Musical Control Of Computers", *Computer Music Journal*, 26(3):11-22.

Wright, M. 1998. "Implementation and Performance Issues with OpenSound Control", *Proceedings of the*

International Computer Music Conference, pages 224-227, October 1998.

Wright, M. & Freed, A. 1997. "OpenSound Control: A New Protocol for Communicating with Sound Synthesizers", *Proceedings of the International Computer Music Conference*, pages 101-104, October 1997.

Zicarelli, D. 2002. "How I Learned To Love a Program That Does Nothing". *Computer Music Journal*, 26(4):44-51, Winter.