CENTER FOR COMPUTER RESEARCH IN MUSIC AND ACOUSTICS
OCTOBER 1993

Department of Music
Report No. STAN-M-84

TOWARDS A BETTER SOFTWARE-DESIGN FOR SUPPORTING CREATIVE MUSICAL ACTIVITY (CMA)

Daniel V. Oppenheim

CCRMA
DEPARTMENT OF MUSIC
Stanford University
Stanford, California 94305

Towards a Better Software-Design for Supporting Creative Musical Activity (CMA)

Dmix

Daniel V. Oppenheim

Center for Computer Research in
Music and Acoustics (CCRMA)

Stanford University 1993

Towards a Better Software-Design for Supporting Creative Musical Activity (CMA)

Daniel V. Oppenheim

Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, CA 94305

Dan@CCRMA.Stanford.Edu

Abstract

It is argued that many current systems for composition of Experimental Computer Music (ECM) do not encourage the composer to explore the full potential embedded within computer-controlled systems for composition. Several fundamental problems that affect the user-interface design seem to have been by and large neglected, and are here carefully examined. The internal representation of music within a computer and the external presentation through which the composer works is discussed, with emphasis on their combined effect on the quality of the user-interface. The user-interface design in several main-stream music systems is critically examined. Some suggestions are made that may lead toward a better design and to enable composers a higher degree of musical expression.

"A major unsolved problem is putting all of the software components together to form a unified music system with a consistent design philosophy." (Roads 85).

Background: Music, Meaning and Expression

Is the King Naked?

During the past three decades the computer has gradually taken its place as one of the most important tools for creating music. Whereas the production of commercial music is heavily dependent on the computer, it had only a limited effect on musical style and none on musical language. The main motivation for using the computer in pop music is to cut down on production time and costs. On the other hand, in art music, analog and digital technology did have a profound and fundamental effect on both musical style and language, and resulted in electroacoustic music. New musical materials unknown during almost 2000 years of the western music tradition suddenly became available to composers. This happened after a breakdown of the classical tonal system, when there was a strong need for a new musical language. It is not surprising that many composers felt computer-music had an outstanding potential, one which could lead to a new era of musical style and expression.

Has this actually happened? There are two observation I would like to make in this respect, and I regret that they are somewhat disturbing.

The first is that many of today's better known composers have indeed experimented with computer music, but eventually gave it up and returned to instrumental music. A few names that immediately jump to mind include Berio, Penderecki, Stockhausen, Crumb and Ligeti (many more could be cited).

A second observation has to do more with style and originality. If the works of the 60s were still experimental and somewhat crude, the mid 70s and early 80s formed a peak of excellent new works which were highly original and innovative. One should note that, during this period, many of these works were still composed using analog devices. During the last decade analog devices have gradually become obsolete and are being replaced by the computer and advanced digital technology. However, it is debatable whether this new digital technology has led composers to discover new musical means of expression that are fundamentally different from, or at least a substantial improvement upon, that which was achieved by the mid 70s using analog or outdated digital technology.

All this may have led to a (false) conclusion that hopes for a new means of musical expression were unfounded, and therefore time spent on the pursuit of this romantic illusion has been wasted. I think it is extremely important to ask '**why** is this so?', or 'is the king (the computer) really naked?'. Several reasons are often given as possible answers: the technology is slow and limited, the learning curve for mastering new technology is prohibitively long, by the time it is mastered it often becomes obsolete, it is hard to match the expressive quality of a live performer, a composer will not readily give up years of mastering musical craftsmanship to become dependent on a 'bad tempered computer', and so on. While these arguments are indeed true, I think they are an inevitable outcome of deeper, more fundamental, problems.

I firmly believe that the computer does indeed possess a wonderful potential for creating new music. Max Mathews once said that "the 'problems' of Computer Music are no longer those of technology but rather of our ability to control it "(Mathews, 1989). This shifts the emphasis from the rationalistic concepts in which we are used to thinking of technology to a much deeper, phenomenological level that should be given careful consideration as a basis for good user-interface design. Systems for creating music were too often designed by engineers who did not have a sufficient understanding of deep musical issues such as: what is music? what is a musical idea? how does a composer express a musical idea? and what is the compositional process? However, not only are these problems extremely complex but one must also realize that there may not be an answer to many of them! Even if an answer were found, there may well be more than one 'correct' answer. On top of that, there will always be a fundamental contradiction between the way humans think about such issues, using non-explicit natural language, and the rational and explicit way in which this knowledge will have to be used by the engineer when designing the user-interface.

The remainder of this presentation will discuss some of the more fundamental problems that affect software design; explain how music is modeled in a computer and the consequences for the user interface; discuss approaches in design taken by several systems; and, will outline some points that may lead toward a better overall design. It should be stressed that this presentation is not a comprehensive scholarly investigation of these topics, though I do hope it will encourage such work. The ideas that follow are based on my experience as both composer and software designer during the past four years at the Stanford's Center for Research in Computer Music and Acoustics (CCRMA). I think that being aware of these problems, and ultimately understanding them, is crucial to the ability to design an environment that is better suited for Creative Musical Activity (CMA).

Some Problems

Conventional Versus Experimental Computer Music (ECM)

In our discussion we shall be referring to two kinds of music: *conventional music* composed using conventional notation and performed by live musicians; and, *Experimental Computer Music* (ECM) that uses new musical materials obtainable only via computer. Many fundamental differences between the creative process when composing conventional music and ECM have been observed (Oppenheim 1986). The conventional composer is well experienced in the use of standardized sound elements and is therefore able to compose directly from a 'musical idea' and to develop it with the aid of notation. In ECM the composer experiences new materials by experimenting with them, and organizes his materials into a work in an interactive and experimental process. In the following discussion we shall try to point out some aspects that are common to both.

Meaning and Context: the Poet Versus the Composer

What is the 'meaning' of a musical idea? Whereas this may not be easily expressed in verbal terms, we understand this intuitively as we listen to the music. Research in artificial intelligence has attempted to understand 'meaning' by creating systems that establish a well defined association between an object whose meaning is sought and another object which explains its meaning. In such systems a given object will always have the same meaning; i.e. meaning is not context dependent.

In music this is not the case. The 'meaning' of a musical idea, motive, or phrase, is highly dependent on the surrounding musical context. The same musical material will often have a different meaning when appearing in different parts of a composition. Moreover, musical 'meaning' is also highly subjective—a listener might find different meanings within a given recording each time he listens to it.

How does all this affect the design of a system for composition? I will demonstrate this with a comparison to a word processor used by a poet as a creative workbench. A poet creates his work by putting words together, and words can be represented as alphanumeric symbols that are displayed on the computer screen. The MEANING of a word, its interpretation, its relationship to other words and ideas—the poetic context—is understood in the poet's head *and is in no way dependent on the computer.* We can safely assume that the computer has a negligible effect on the poet's ability to understand the meaning of his poetic ideas, or to develop them into a work of art. If the poet types the word 'dog', then whether it is meant as a feeling of admiration towards man's best friend or used as a curse word, is NOT affected by the computer system, by the type and size of font, or by the color on display.

On the other hand, this is not the situation in ECM. A musical idea remains MEANINGLESS unless the composer can make it concrete by synthesizing it. This can only be accomplished via a computer system. The process of composition can only begin once a musical idea is made concrete. This initiates a cyclic process of experimentation and interaction which is greatly dependent on the system and affected by it. Whereas a word processor cannot distort a poet's idea, a musical idea often gets distorted in the process of making it concrete. Whereas the context of a poem is clear in the poet's mind, musical ideas must be experienced within the overall music-context before it can be determined whether the composer's intentions have been conveyed. And whereas the word processor is a technical instrument used for convenience, the composer's ECM environment is the ONLY means available for discovering new ideas, experiencing them, experimenting with them and gradually organizing them into a composition.

The design of a good interface for ECM is therefore essential to enable Creative Musical Activity (CMA). A 'bad' interface will have a direct effect on the composer's ability to express his musical ideas, to fully grasp their meaning within the overall musical context, and hence on the QUALITY of his artistic expression! In this respect, the main problem that the user-interface must deal with could be stated as: "how do we enable composers to express their abstract ideas using a formal system that has little to do with their musical, intuitive and informal way of thinking?"

Composition and Performance

Creating music in the Classical period was a two stage process: the composer would first create a score, and then a performer would perform it. Where is the 'music'—in the score or in the performance? An acceptable answer might be—in neither, and yet in the combination of both. The traditional score determines melody, harmony, structure and so on. But 'music' is much more than that. The performer adds the musical nuances without which it would sound mechanical and 'unmusical'.

This point seems to have been overlooked by many designers of systems for composition. Performance in the domain of computer-music should not be limited to the notion of improvisation, score tracking, or interactive settings (even though these are extremely important aspects). If the composer is creating a computer piece, he not only creates what is analogous to the classical score, but must also add the equivalent of the performer's musical nuances.

With this in mind it makes sense to consider that tools normally used only by a performer are just as useful to the composer. While creating it is common for a composer to sketch out a musical idea only to find out that it doesn't work—not necessarily because his idea was 'wrong', but because there were no tools that enabled him to shape the phrasing, dynamics, gestures, etc., and add the musical-nuances that will transform his idea into a musical entity.

Rationalism Versus Phenomenology

To further emphasize the problems stated in the last two paragraphs let's examine the way musical ideas are currently expressed in computers. In many cases they are expressed in the form of a notelist, or an algorithm that produces a notelist. One must ask once again—what is a musical idea? can it be formalized and quantified into a list of parameters?

This is where observing a score of conventional music can be misleading. It is tempting to assume that as a score is equivalent to a notelist, the latter is a valid means for musical expression in ECM. One can easily locate within a score a phrase or motive that encapsulates a musical idea. However, the notation is NOT an accurate representation of the music, but is more a form of tablature that tells the performer how to play. The music itself is far too complex to be accurately represented in a score. Expressing musical ideas in a score works well in conventional music as the performer will add all that is missing. But this method can be limiting in ECM when using notelists. Whereas any musical work can be analyzed and transformed into a list of parameters, *this process is not reversible*. To expect that the composer will always be able to find the set of parameters which would express his musical ideas is as naive as believing that the knowledge of the composition of every molecule that exists in a human body permits the synthesis of life!

The misconception noted above stems from a long tradition of rationalistic thinking that has been embedded in our western tradition since the days of the early Greek philosophers. We take it for granted that 1 always equals 1 (1=1). Computers are programmed with formal languages; if computers would ever indicate that a 1 does not equal another 1, they would probably be declared unusable. With all this well rooted in our background, describing musical ideas as a list of parameters seems to make sense. However, the way we perceive phenomena is NOT rationalistic. A musical idea might be perceived differently, and therefore have a different meaning, as it appears in different sections of the same composition. All this must be taken into account when designing a user interface, and alternatives that are better suited to our natural way of thinking must be provided (Winograd 86).

Other Domains of Activity in ECM

Composing ECM is a complex process. The traditional approach to system design has been to break down the overall task into well defined activities, and handle each activity with a separate application. Figure 1 outlines the main groups of activities: composition, performance, synthesis, editing and visualization (it may be worthwhile adding psychoacoustics, analysis and archiving). This segregation is artificial and is the cause of severe limitations in existing systems.

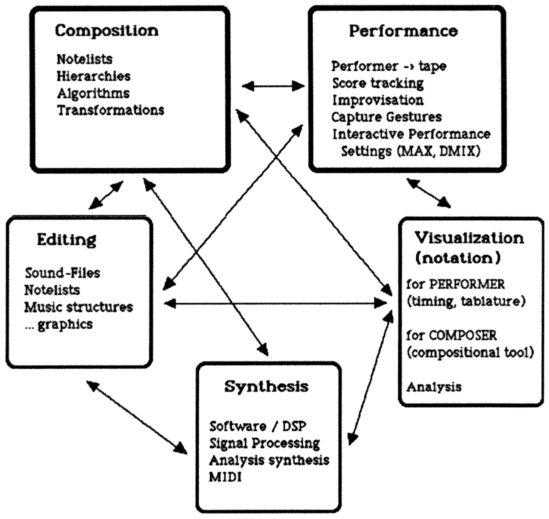

Figure 1

The previous section already demonstrated that there is much to be gained by encapsulating composition and performance into a single environment (see also Oppenheim 1991). The same reasoning can be applied for the remaining domains, as they are all closely interlinked. For example, in Music-N and Midi systems, synthesis is a separate activity that precedes composition. This works well with music that is essentially conventional, as that music emphasizes pitch content rather than timbre. But this makes it extremely difficult to work with a single sound (note) that evolves and changes over time, as might be suggested by some works of *musique concrète*. Similarly, editing techniques used for mixing sound files may be just as applicable to notelists that represent sections of a composition. Signal processing techniques used for processing sound may, under certain conditions, generate interesting musical results if applied to sections of a composition. The arrows in figure 1 represent many possible, and indeed desirable, relationships. The encapsulation of all these components into a single environment with a unified user-interface would not only simplify the overall use of the system, but may also lead to the discovery of new musical possibilities.

The design of a single system that will encapsulate all this functionality is highly complex. However, recent developments in object-oriented technology do suggest that it is feasible.

Opposites: Which is the More Desirable?

There are many different approaches within computer music, many of which are often considered opposites. Consider the following:

off-line	<->	real-time
low-level	<->	high-level
algorithms	<->	improvisation
note-lists	<->	graphics
programmable	<->	user-friendly

What is preferable: composing in real-time or off-line? Are the results produced by algorithms more musical than those made by improvisation? There is, of course, no correct answer to such questions. The choice between one or the other is indicative of a personal aesthetic. Moreover, a composer might use both for handling different musical situations. It is therefore highly desirable that a system provide the composer with all these alternatives to choose from. Yet, it is amazing how many existing systems enable only one of the two!

Perception Versus Parameters

The lack of a clear correlation between the physical parameters that are used by a computer to synthesize music (frequency, amplitude, etc.) and the perceptual parameters with which the composer thinks (pitch, loudness, etc.) is well known. Moreover, the same physical sound may be perceived differently when heard in a different musical context. As a result, it is difficult to predict the sound that will be perceived by a new set of sound parameters. The user interface must therefore support an interactive and experimental working process (Oppenheim 1986a).

Modeling music with computers: the User Interface

Representation and Presentation

Music can be manipulated by a computer only if it is represented within it. This internal REPRESENTATION is implemented in a formal computer language. It is by now obvious that this internal representation will be fundamentally different from the way the user perceives the same music. We shall refer to this difference as a distortion.

The user manipulates the music via the user-interface. This interface is a PRESENTATION of the internal representation that may consist of graphics, text, keyboards, etc. The actions carried out by the composer when manipulating the user-interface affect the internal representation within the computer, and that in turn controls the synthesis of music. The QUALITY of a user interface can be evaluated by the amount of distortion that occurs between the music the composer intended to produce and the music that was actually produced.

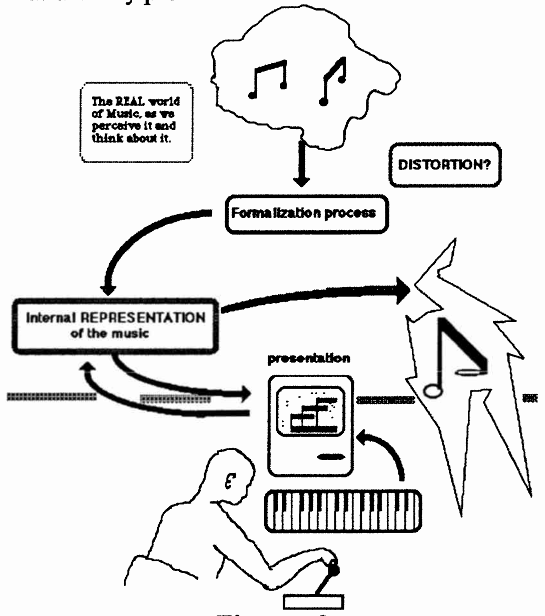
The REAL world of Music, as we perceive it and think about it.
DISTORTION?
Formalization process
Internal REPRESENTATION of the music
presentation

Figure 2

The user-interface should be MEANINGFUL—it should allow the composer to manipulate it while thinking in his own, familiar, musical-concepts. The user interface may hide the internal representation from the user, under the assumption that this formal representation will not help the composer, or it may take an opposite approach and highlight some of its aspects and enable the user to control low-level elements of them.

Evaluating the User-Interface Design

In evaluating the design of a user-interface a very general approach will be taken. We divide the various mechanisms with which a user can control a system into four categories: Programming languages, high-level tools, graphics, and gestural input.

High level **programming languages**, such as C, Lisp or Smalltalk, are found in systems that attempt to provide the user with 'unlimited' control, but require a knowledgeable user to take full advantage of their capabilities.

High-level **tools** perform a specific task and are easier to use, but because they are high-level they are less general and less flexible than programming languages. For example, Common Music provides the 'item-stream' mechanism for handling lists of parameters (Taube 91). Some tools cannot be modified, such as in commercial MIDI sequencers, whilst other may be modified by the user, as in PLA (Schottstaedt 84) and Dmix (Oppenheim 89).

Graphics provide visual feed back and an intuitive way to manipulate music, but are not as general as programming languages.

Gestural input is a category that includes an ability to interact with a performer and control the system in real-time via some hardware interface.

The user-interface will be evaluated according to the following criteria:

1. What categories are available—the more the better.
2. If more than one is available, can one category interact with the other and extend it, or is each treated as a separate and independent entity. Interaction between modules can make a system much more general and flexible.
3. Is the internal representation hidden from the user, or is it accessible. Allowing the user to examine and manipulate the internal representation improves flexibility.
4. Can the user modify the internal representation. This is an important point. Since the internal representation will have a considerable effect on the amount of overall distortion produced by the interface, the composer's ability to define a representation that is close to the way he is thinking of the music may help reduce this distortion.

Approaches to System Design

The design of four typical systems will be discussed: a commercial Midi sequencer, PLA and Common Music, MAX (Puckette 88), and Dmix (Oppenheim 89).

Commercial MIDI Sequencers

The design of commercial sequencers is extremely limiting. Music is handled primarily through a graphic interface and notelist editor. The user-interface presents a metaphor taken from the analog recording studio of Midi events and tracks. Tools allow operations such as cut, copy and paste. The internal presentation is hidden from the user, not accessible and cannot be changed.

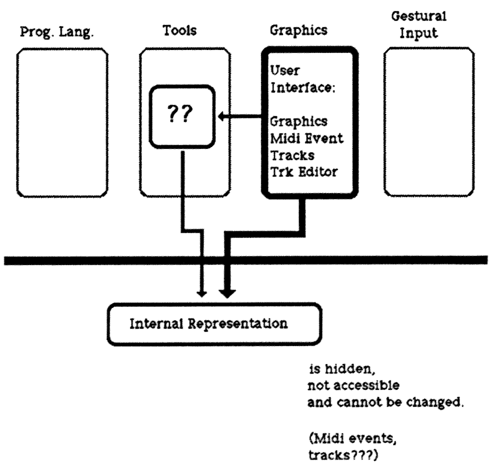

Figure 3

High Level Languages: PLA and Common Music

PLA and Common Music represent an approach to design that, for many years, was the ultimate in power, flexibility, modularity and extensibility (Schottstaedt 84, Taube 91). They provide powerful tools that enable the composer to work without having to become overly involved with programming. The two way relationship between the tools and the programing language is one of the most powerful features of their design (figure 4). This ensures that existing tools can be modified to handle special musical requirements, and that new tools may be programmed by the user if needed. The internal representation can be thought of as a notelist in a text file; it is transparent, though not readily accessible from within the environment.

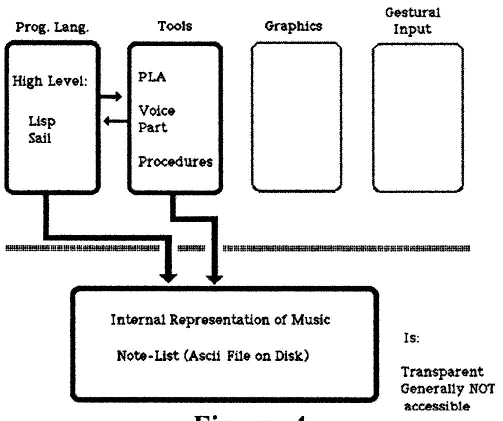

Figure 4

MAX

MAX is designed for real-time performance and is probably the finest example of a graphic programming language. Its design is extremely well thought out—notice connections between all four categories (figure 5). As MAX is programmed by manipulating graphic icons, musicians can quickly and intuitively create complex performance settings without having to spend time learning programming. A performer's MIDI input gets processed in patches programmed by the user, and accordingly MAX generates MIDI information in real-time. MAX was designed only as a performance tool; therefore, the music it produces cannot be captured and used off-line within MAX as compositional material.

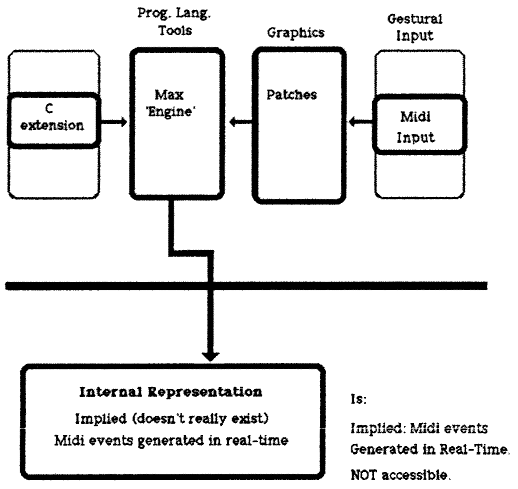

Figure 5

DMIX

Dmix is an object-oriented system designed with the intention of implementing many of the ideas that have been brought up in this presentation (Oppenheim 89). Figure 6 demonstrates clearly the intricate interrelationships existing not only between the four main components of the user-interface, but also between the interface and the underlying internal representation. Moreover, these relationship are by and large a two way path. For example, MAX-like objects not only facilitate interaction with a live performer, but also capture performance gestures that may be worked into a composition. These gestures may also be used to add performance nuances to already composed sections (Oppenheim 91). The functions of graphic editors can be modified and extended by using high-level tools with which the composer is already familiar; algorithmic music-generating objects can interact with graphic editors, or even with the real-time, MAX-like, objects.

A basic design philosophy has been to make available many different, even contrasting, tools for manipulating music (assuming that each will be useful under certain musical conditions) and to enable the composer to switch between tools at any time. This can become very confusing unless all tools share a unified user-interface. This simplicity was by and large made possible by making the internal representation MEANINGFUL to the user. The internal representation is a collection of objects that model music by using the concepts with which a composer is familiar, such as notes, phrases or sound files. This representation is made completely transparent to the user so that he may easily modify or extend it, and model his own way of musical thinking.

Moreover, not only can the underlying representation be manipulated by the higher level user-interface components, but the representation can also be transformed into some of the tools that are used for manipulating music. Whereas this may sound confusing, in reality the two-way interconnection between the internal representation and the user interface is simple to understand and easy to employ.

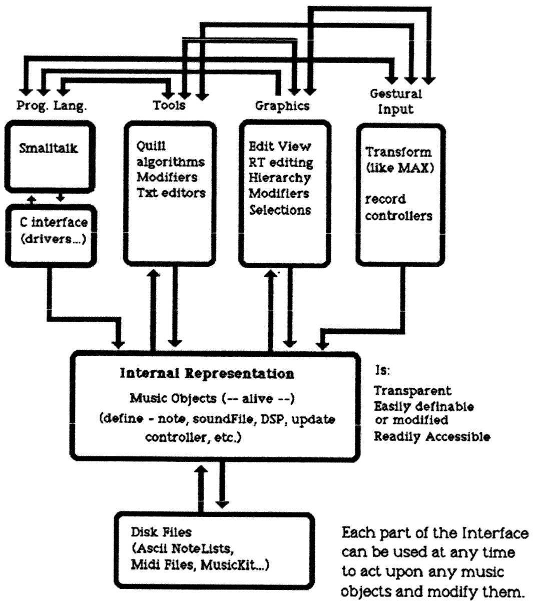
Prog. Lang.
Tools
Graphics
Gestural Input
Smalltalk
C interface (drivers...)
Quill algorithms Modifiers Txt editors
Edit View RT editing Hierarchy Modifiers Selections
Transform (like MAX) record controllers
Internal Representation
Music Objects (-- alive --)
(define - note, soundFile, DSP, update controller, etc.)
Is: Transparent Easily definable or modified Readily Accessible
Disk Files (Ascii NoteLists, Midi Files, MusicKit...)
Each part of the Interface can be used at any time to act upon any music objects and modify them.

Figure 6

Conclusions

The problems which have to be solved in order that computers may better support creative musical activity (CMA) are conceptual problems: design, rather than advances in technology. Phenomenology, rather than rationalism, seems a better approach for understanding these problems (Winograd 86). While the dichotomy between the formal mechanism that underlies the way computers operate and the way composers perceive musical phenomena will always prevail, ways around it that minimize its adverse affect on CMA can be found.

Many excellent tools for creating music have been developed during the past three decades of computer music. It would be nice if a way were found for composers to express and develop musical ideas intuitively and without having to deal with the computer—however, it seems unlikely this will happen in the near future. Rather than looking for new ways, we are confident that existing techniques can be very productive if embedded in a single environment with a simple and unified user-interface.

No reference was made in this presentation to such important concepts as modularity, extensibility, multiple presentations, or user friendliness. We view these issues to be of secondary importance in comparison to the conceptual issues we raised, and in the implementation of a good design they will most probably take care of themselves.

References

Mathews, M. 1989. private communications.

Oppenheim, D. 1986. "The Need for Essential Improvements in the Machine-Composer interface used for the Composition of Electroacoustic Computer Music," *Proceedings of the ICMC*, the Hague.

Oppenheim, D. 1989. "Dmix: An Environment for Composition," *Proceedings of the ICMC*, Columbus, Ohio.

Oppenheim, D. 1991. "SHADOW: An Object-Oriented System performance system within DMIX," *Proceedings of the ICMC*, Montreal, Canada.

Puckette, M. 1988. "The Patcher," *Proceedings of the ICMC*, Cologne.

Roads, C. and John. S., editors, 1985. "Foundations of Computer Music," MIT press, Massachusetts.

Schottstaedt, B. 1984. "PLA - A Tutorial and Reference Manual," CCRMA report No. STAN-M-24, Department of Music, Stanford University.

Taube H. (1991) "Common Music: A Music Composition Language in Common-Lisp and Clos," CMJ 15/2. MIT press, Cambridge, Massachusetts.

Winograd, T. and Flores, F. 1986. "Understanding Computers and Cognition—A New Foundation for Design," Norwood, NJ: Ablex.