DMIX - A MULTI FACETED ENVIRONMENT
FOR COMPOSING AND PERFORMING COMPUTER MUSIC:
ITS DESIGN, PHILOSOPHY, AND IMPLEMENTATION

Daniel V. Oppenheim
DMIX—A Multi Faceted Environment for Composing and Performing Computer Music: its Design, Philosophy, and Implementation

Daniel V. Oppenheim

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

Dan@CCRMA.Stanford.Edu

Musical creativity is a process about which little is known. Yet the most important function of a system for aiding music composition is to support this process. Taking a fresh approach, that is more phenomenological and less formalistic, a new software design that exhibits several desired characteristics missing in many existing systems was implemented in DMIX. This presentation explains the rational and design of DMIX—an environment dedicated to composing and performing music.

"... no one has yet figured out how to harness this expressive power effectively. We observe instead that research is preoccupied with identifying and refining the suitable computable formalisms for music." Loy, 1989.

Prelude: Music and Creativity

Music is the most abstract of all art forms, and for me, the most expressive. With over a thousand years of musical tradition, Western composers have crafted works of fascinating beauty, infinite expression, immense magnitude, and astonishing sophistication that can be found in counterpoint, harmony, motivic development, orchestration and form. Technology today opens new horizons for both musical expression and materials. However, does technology also expand the horizon of our creativity? Does it effect our creativity? What is the relation between the media and our creativity? In other words: are we making full use of the potential that technology has to offer?

These are questions that I have been asking myself for the past decade. I believe that we are exploiting only a small fraction of the potential that technology can offer. My feeling is that the technical problems will sort themselves out: what is slow today will be realtime tomorrow. We are now in a position to tackle much deeper problems of a conceptual nature, problems that cannot easily be defined in mathematical terms, but problems that directly effect our creative and expressive abilities. Being a composer, my research is focused on trying to understand these problems and find practical ways in which to improve our ability to be creative while using technology. Specifically, I am working on software user interface, human computer interaction and system design.

Any discussion of creativity, the creative process, the effects of technology on our creativity, or how technology can better adapt to support creativity, is problematic at best. It is hard, if not impossible, to define what creativity is—so how can we design technology that will support it? And if we think of music—what is music? what is a musical idea? is a musical idea independent of its context? Non of the above can be formalized into a neat set of parameters that can then be fed into a computer and produce acceptable results. On the other hand, we intuitively (naturally) understand a lot of the above. They become unclear and vague only when we try to explain them verbally or in formal terms that contradict their very nature.

Software design

Software design typically consists of two stages: first, the task domain is modeled and formally represented in the computer in the form of data structures and the functions that manipulate these structures. This is called the internal representation. Then, the task itself is modeled in order to determine the software design. Now the application can be built on top of the internal representation. Let us look at a concrete example from the sciences. In designing software to control a space shuttle, the internal representation would include things such as gravitational forces, mass, trajectories, engine characteristics and so on. Then, the software design would allow the plotting of a flight course, generating engine control signals, flight navigation, life support, and alike. In the case of
the space shuttle things are so well defined that flight control is by and large automatic.

With music things are not that clear. Since there is no formal understanding of what music is, it is not possible to come up with an internal representation that will be general enough for all composers in all domains of computer music. Since we do not fully understand the creative process (or even what a composer does while he composes) it is not possible to come up with a software design that would readily support this process. Furthermore, not only do individual composers work differently, but they often change their techniques from piece to piece, or even while working on a single piece. A 'good' system for composition must be extremely flexible and allow users not only to define and/or modify its internal representation, but also to reconfigure its design so that it best meets their individual needs.

Let us make a comparison by examining more closely what happens in the well defined system of a space shuttle. Problems arise when malfunctions that were not considered during the design stage occur, as these cannot be handled by the system. In such situations, or breakdowns as termed by the philosopher Heideger, the astronaut must be CREATIVE and find new solutions. His survival may well depend on the system being sufficiently flexible to perform these new tasks for which it was not designed. Here, as in music systems, the key to success is in supporting CREATIVITY. The space shuttle may not support an astronaut's arbitrary (creative) actions, as those are not part of the software model. But in music we must begin by supporting the individual's actions—a system that will not allow a composer to be creative is unacceptable. It seems that a system design supporting Creative Musical Activities (i.e. CREATIVITY) begins in the gray area in which other applications end: at the stage of the breakdown.

Existing Music Systems

Gareth Loy in his excellent article "Composing with Computers" (1989) differentiates between four types of systems:

- for music data input (such as Music N [Mathews 69] and SCORE [Smith 74])
- for editing music via textual, graphic or sonic representations
- for specifying compositional algorithms (such as PLA [Schottstaedt 84])
- for automated composition via program input (such as Compose [Ames 90])

For the sake of completeness I would like to add:

- for performing music (such as MAX [Puckette 88])

Many wonderful compositions have been created using these systems. However, I believe that today we are in a position to point out some aspects that constrain the creative process. These points are brought up because they play an important role in the design and implementation of DMIX, as will be discussed later.

1. A single user interface metaphor. For example, in data-input systems (Music-N) music can only be created by writing notelist in a text editor; in PLA and Common Music music can only be expressed by writing short computer programs (algorithms); in commercial sequencers music can be input in realtime or via graphic manipulation but cannot be manipulated on a high level using conventional programming techniques; MAX enables realtime interaction with a performer but does not readily lend itself to carefully working out and refining precomposed sections. Clearly, one metaphor is not better than another—they are all useful under certain conditions and should all be equally available to the composer.

2. A limited view of the creative process is implied by many of these systems. For example, in PLA and Common Music composition is viewed as an off-line (non realtime) process in which the composer gradually builds a score by writing small programs (the PLA voice); editors and many sequencers model composition as the mixing of musical materials to form larger sections of music; in MAX music is created by defining algorithms (patches) that respond to the performer's gestures in real time.

Clearly, all these approaches are valid but they are only a part of what composing is really about: Can we segregate composition and performance? Does the composer of computer music not also perform his music; and, therefore should a system not include tools for performance as an integral part of its compositional tools? Is the best way always an off line process or is it always a real-time process? Whatever this process is, it is clear that these two extremes are but parts of it, and that a system should support both equally well.

3. The limitations of the score model. Almost all existing systems use the score, or eventlist, as a model and end goal of composition; most compositional tools either create or manipulate scores. In Classical music the end product of composition was indeed a score. However, the Classical score is NOT a
representation of the music, though the score does represent compositional ideas, structure, and so on. The score is merely a form of tablature that enables the performer to play the piece. The MUSICAL OUTCOME is the COMBINATION of the composer's compositional ideas and the performers musical nuances, articulations, and gestures.

In computer music, especially in works that do not involve a live performer, the composer must also 'perform' his work. There must be tools that will enable the composer not only to define the score—the compositional ideas and structures—but also to perform it and give life to his abstract musical ideas while adding the infinite details that a performer normally would. This is extremely problematic. Whereas there are many well defined formalisms for composing and manipulating music on the level of a score, there are none for dealing with performance practice.

As Loy points out: "research is preoccupied with identifying and refining the suitable computable formalisms for music." By focusing on a single formalism these systems become closed systems. Like the space shuttle, such systems work well as long as they are used in accordance with the model they are formalized on. But humans in general, and composers in particular, often have a crave for exploring the unknown...

Other problems: Perception, Concepts and Meaning

There are several other fundamental problems that are characteristic of the medium of computer music and have a profound effect on the creative process. These have been well documented but do deserve a brief mention (Schaeffner 67, Oppenheim 86, 91b). The first is the lack of a simple correlation between the way in which we perceive sound and the physical parameters that must be specified to synthesize it. This phenomena is really a part of a much deeper dichotomy: the NATURAL way we think about music and form our musical CONCEPTS as opposed to the FORMAL way in which we must manipulate the computer in order to realize our creative ideas. There is no simple way of doing this (Artificial Intelligence has not yet succeeded in solving this problem).

A second domain falls in the realm of MEANING. First, a musical idea is irrelevant unless it can be produced by a system and made concrete. Often, for reasons explained above, the idea will get distorted by the system during the process of materializing it. Therefore, the composition, the creative process, is not only dependent on the specific system that is being used but is also significantly affected by it. Second, the meaning of a musical idea is highly dependent on the context in which it is embedded. In fact, the same idea, or same musical material can have DIFFERENT meanings as it appears in different sections of the work. One cannot work on a section of a composition in isolation from the whole.

All this brings out the importance of experimentation, feed back, interaction, and context orientation as vital characteristics of a system for composition. The way in which these are implemented will have a significant effect on the creative process.

Fuga: The DMIX Project

Exposition: The user interface, Goals, and Design issues

DMIX is an environment dedicated to composing and performing music. I have been designing and implementing DMIX over the past several years at the Center for Computer Research in Music and Acoustics (CCRMA) in Stanford University. DMIX is an object oriented system implemented in Smalltalk-80/Objectworks (see Pope 89) that currently runs on a Macintosh II computer, but will soon run also on NeXT, IBM, Sun, Silicon Graphics, Atari, and other platforms. It is an ongoing project and I expect significant changes and improvements to take place as my understanding of the problems and experience increase.

My main motivation was to design an easy-to-use and yet flexible and general environment that has a uniform user-interface, provides multiple representation, is easily extendible, and is independent of any synthesis hardware or host computer (see Oppenheim 89, 90, 91a). An important part of the Design philosophy was to view all activities associated with computer music as part of a whole. In particular I am trying to blend the activities of composing and performing, the concept of real-time alongside off line, the notion of algorithmic juxtaposed with improvised, alphanumeric programming as an extension of graphic manipulation, and so on.

A major goal in designing the user interface was to enable composers to work with minimum interruption of the creative process; once a musical idea has formed it should be easy to find a way to implement it; during implementation the composer should not have to spend time writing code or consulting operating manuals. Thus, the focus is on the ability to be CREATIVE—on providing the means to easily express musical ideas, to experiment with them, and to gradually organize them into a musical composition.
DMIX is unique in several respects. To my knowledge it is the first environment that directly addresses issues of creativity and enables the composer to interact with his musical ideas while communicating with the full context of his music. This is achieved by simultaneously having different ways of visualizing and manipulating the music on every level: from the lowest level of the sonic event to the highest level of the composition’s hierarchy. It is also a very extensive system that offers a rich variety of tools for creating, editing, modifying and performing music. With a large palette of tools (that are often redundant and overlap in functionality) composers have a better chance at finding a tool that best suits their individual way of thinking. However, what makes DMIX unique is not any specific tool, but rather the way in which:

- tools can interact with each other (a property I term slappiness)
- tools can be grouped and reconfigured to form new ways of interacting with the system
- tools can transform into music and vice versa

I find these properties of paramount importance in bridging the gap between the composers CONCEPTUAL way of thinking and his ability to bring his concepts to life with the aid of a computer. If musical material can be transformed into tools that can then operate on other musical materials and vice versa, then these tools are no longer abstract or formal operators, but can rather be conceived in musical terms. For example, the composer can think in terms of: “lets have the rhythm from this section determine the harmonic changes of that section ...”

The ability to modify, reconfigure and group existing tools into new tools, is a high level approach to modularity. This enables composers to develop their own tools and hence their own ways in which they can think of and CONCEPTUALIZE their musical ideas.

**Episode: Schematic System Overview**

Following is a very brief description of the main elements of the DMIX system. The tools can be roughly divided into six groups, though many tools overlap several categories:

- graphic based
- text based
- real time
- performance (for composition or performance)
- general tools (functions, patterns, ...)
- the music representation

**Graphic based tools**

Graphic editors provide an intuitive means to manipulate music. Several mechanisms expand the capabilities of graphic manipulation:

**REAL-TIME-EDITORS** enable editing music as it is playing and provide visual feedback. Through this mechanism the composer can intuitively perform the changes that are ‘right’ in respect to the overall musical context.

**SELECTIONS** group events into higher-level units that can be manipulated as a whole.
**MODIFIERS** facilitate for intuitive high-level operations. Functions can be used to determine tempo maps, set or scale parameters, determine temporal and rhythm characteristics, and the likes. Filters can control pitch content, mode, or harmony. Interpolators can nest other Modifiers to form more complex Modifiers. The way in which a Modifier functions is defined in a CodeDictionary—the user can easily change a Modifier's functionality and add new modes of operations. Modifiers can be created from musical material and *vice versa*.

In the following example the Bach Prelude in C major is 'Jazzed up' using the rhythm from an improvised melody. A Modifier is created from the melody and then applied to the Prelude.

**CODE-DICTIONARIES** apply blocks of user-defined Smalltalk code directly to the graphics. This mechanism infinitely extends the capabilities of graphic editing and blends graphic manipulation with programming. When using Code Dictionaries all aspects of the graphics and windowing systems, as well as the underlying music-object structure, are already handled by the system—the user need specify only the changes that he wants to take place.

Using Modifiers to map musical qualities from one event to another

We can now see the larger picture: the functionality of graphic editing is significantly enhanced by using Selections, CodeDictionaries and Modifiers.
hope that this tool will help in visually grasping the overall musical context, and will become a compositional tool much like the conventional score that aids composers in working out countermelodies, thematic development, and large forms.

**text based tools**

**QUILL** is a high level algorithmic music description language (Oppenheim 1990). It is modeled after PLA and Common Music (Schootstaedt 1986; Taube 1990). Yet it offers an interactive environment that integrates well with other tools in DMIX.

**Hierarchical views** manipulate large sections of a composition with all their sub parts.

**PGG** (Personalizable Graphic Generator)—yet to be implemented—provides an extended graphic visualization of the music in a perceptually meaningful way (Oppenheim 87). That is to say that the physical parameters used for synthesis are mapped onto the perceptual sound qualities that the composer is concerned with (as opposed to a mere combination of pitch, amplitude and/or spectral presentations). It is my
DMIX

Tools for Realtime

INPUT is a class that connects to external hardware input devices, such as MIDI sliders and wheels. Inputs are used to map user actions in realtime, as in realtime editing. In a future implementation the user will be able to freely connect Inputs with other objects in DMIX.

ECHO is a collection of classes that represent processes that interact in realtime with a performer (or the composer). In many ways an ECHO resembles a patch in MAX. Interactive sessions using ECHO can be captured in DMIX for further processing (with QUILL, Graphic editors, etc.). ECHO processes can be spawned and controlled via SHADOW in the context of a live performance.

The Performance system comprising of SHADOW (the tracker) and a number of Echoes

LENNY is a tool still under development. Its objective is to enable a composer to gradually refine his music by adding the performance nuances normally added by a live performer.

Performance Tools

SHADOW is a system for both composing and performing interactive music. The composer creates a score via various graphic and text editors. The score links triggers—notes or performance gestures—to specific actions. These actions can be the playing of additional notes or sequences (i.e., an accompaniment) or executing blocks of Smalltalk code. The latter can be used to spawn and update ECHO processes in order to facilitate for what I term non-structured interactive tracking. During the composition stage SHADOW can emulate the performer and allow the composer to play the score and test out interactive sections.

During a performance SHADOW follows the performer, synchronizes with him, and executes the predefined actions in the score while taking into account the performer’s tempo. Tracking can be any combination of linear or interactive. Linear tracking is the more conventional approach where all the triggers are predefined sequentially in the score. In interactive tracking scenes are specified, each having several triggers. Here the performer determines what triggers he will play, how many times he will repeat each trigger, and in what order. This can enable the performer to expressively control new musical parameters, such as timbre, texture, harmony, or even what music will accompany him.

General Tools

MODIFIERS include Functions, Filters and Interpolators and have already been described in relation to graphic views. However, Modifiers can be used anywhere within the DMIX environment, including in QUILL where they can aid algorithmic composition, Smalltalk inspectors, and more. A special feature in DMIX is that Music-Events can be transformed into MODIFIERS that can, in turn, be applied to other music objects, and vice versa. This offers composers alternative ways of thinking about transformations of musical materials and may lead to new approaches in dealing with musical concepts such as motivic treatment and development.

Patterns are list processing elements modeled after the Common Music Item List (Taube 1990). They are part of a large package of objects that can be used for algorithmic composition, statistic distributions, and the likes.

There are many additional tools within DMIX that cannot be described within the limited scope of this paper.

The Music Representation
I consider the internal music representation, i.e. the low level Smalltalk objects on which DMIX is built, as a user-accessible tool. This a good design feature: a visible and accessible internal representation allows users to modify, extend, and adapt the entire system to handle the new musical situation that they conceive. Tools in DMIX are designed to keep on functioning even when changes are made to the internal representation (within reasonable limits).

Coda: The Big Picture and Slapability

We can now examine several ways in which the components in DMIX function together and form a unified music system with a consistent user interface.

Multi presentations

DMIX readily supports multi presentations. The following diagram illustrates viewing the Bach Prelude via a graphic editor, notetlist editor, and Smalltalk inspector; other presentations are available. Changes made in one view will reflect in all others.

Using all the tools at once

The following diagram demonstrates the ease and flexibility of bouncing music back and forth between different tools. Here we are moving between the extremes of real-time improvisation and off-line algorithmic processing (alphanumeric). Material was generated in real-time by improvising with an Echo. The music was captured in DMIX and then edited graphically. At the next stage a NoteList editor was spawned to enable detailed editing that might be hard to do via graphics. The text representing the music was then copied into the QUIL editor, where the music was processed further via high-level algorithmic programming techniques. This sort of work-process could go on indefinitely.

Slapping one tool onto another

A much more intuitive, powerful, and flexible mechanism for working with several tools or moving between them is what I term Slapability. In the previous example the composer had to record his action using the Echo, and then spawn a graphic editor on the completed recording (event). In order to open the TextEditor the composer sent a message with that request to the event (via a menu selection). Text could be copied from the TextEditor into QUIL because QUIL understands the format used by the TextEditor.

Slapping offers a much more straightforward way to accomplish all this and much more. The idea is that by dragging one view and dropping it on another some action takes place. For example, slapping a graphic view on an Echo will cause the graphic view to become the input to that Echo (see next figure).
Dropping a graphic view, NoteList, or a music event, on a QUILL window, will transform that music-event into an alphanumeric algorithmic format that when evaluated will create an identical music-event (see next figure, simplified). However, by changing this algorithm new kinds of musical transformations can easily be made. Slapping a Modifier on a Graphic view will modify the events on the display. Slapping a music event on a Modifier changes that Modifier so that its data is derived from that event. Almost all the objects and views in Dmix can be slapped on each other, providing a fast and intuitive way of transforming musical material.

In a future release I would like to extend the notion of Slapability and combine it with modularity. The idea is that music-events, tools, processes, blocks of Smalltalk code, Modifiers, and any other object could be linked together to form some higher order entity. This is not unlike connecting boxes in the MAX environment. However, here the contents of each box could be a very high-level object or process. This could help composers in finding ways to build new tools for creating and manipulating their music. If the composer is also the creator of his tools, there is a better possibility that he will be able to conceive the music he is creating in terms of the tools that can produce it. This might help to further bridge the gap between the composer’s CONCEPTS and the FORMAL steps needed to realize them with the aid of a computer.

Postlude

DMIX is a first step in the design of environments that better support creativity. Whereas DMIX opens new conceptual ways for thinking about music and about the tools to create and manipulate it, there is still much to be desired. It is interesting to observe that whereas computers open a vast space for new creativity and expression, the composer’s ability to freely express his ideas and work them out is still constrained. Until better user interfaces are found, it will be the conventional composer that enjoys the most creative freedom as he sits by his desk with a pencil in his hand and a score lying before him...
The design process is not unlike composing experimental computer music: as experience in using the system is gained, the insight insight needed to refine the design and improve it will be obtained.

Availability

Musicians interested in using DMIX are encouraged to contact the author.

References


