Barry Truax
Department of Communication and
Centre for the Arts
Simon Fraser University,
Burnaby, British Columbia
Canada

The PODX System: Interactive Compositional Software for the DMX-1000

Introduction

The DMX-1000 is a high-speed, microprogrammable signal processor marketed by Digital Music Systems, Inc. (Wallraff 1979). It is designed to be interfaced to and controlled by a Digital Equipment Corporation (DEC) PDP-11 or LSI-11 computer, and because of its computational speed (up to five million instructions per second), it can be programmed to provide polyphonic real-time synthesis or digital signal processing. The software package that can be purchased along with the machine, called Music-1000, resembles Music 11 (Vercoe 1980) and is a general-purpose synthesis and score-processing language. Although Music-1000 performs scores in real time, a considerable time is required for preprocessing (compiling) the score and orchestra. Moreover, scores are entered and modified through a text editor and no higher-level compositional programs are available within the system.

The PODX system, on the other hand, is a collection of compositional programs that exploit the interactive potential of the DMX-1000. Each program implements a model of interactive composition that allows the user to design sounds and/or structures with immediate aural feedback of the synthesized results within the real-time limits of the DMX-1000. A range of compositional procedures is available and is constantly being expanded and upgraded as the experience of a group of composer users suggests new directions. Although the principal synthesis model in use at present is frequency modulation (FM) (Chowning 1973), the system is currently expanding to include waveshaping and additive synthesis. The microprogrammability of the DMX-1000 ensures that any number of synthesis algorithms, presently known or about to appear in the future, can be introduced and refined within the basic structure of the system.

Computer Music Journal, Vol. 9, No. 1,
Spring 1985, 0148-9267/85/010029-10 $04.00/0,

Interactive Composition

Computer music composition systems can be classified most generally by where they fall along a continuum between computer-realized composition and automated systems. In computer-realized composition the machine processes a score and realizes the sound result without having participated in the actual compositional process. In automated systems most, if not all, of the compositional process is carried out by programmed algorithms. Music V types of programs (Mathews 1969) tend to be of the former category, and programs such as Koenig's Project 1 and 2 tend to be of the latter (Koenig 1970a, 1970b). This range of programs can also be characterized in terms of the concepts of generality and strength (Truax 1980). Systems for computer-realized composition achieve their generality of output with a corresponding loss of efficiency (or strength) by which that output may be achieved. In such a system, it usually takes about as long to generate a simple result as it does a complex one, since general methods are used for both. In addition, to achieve a wide range of output, the user must furnish a great deal of data describing it, since few automated procedures are available. In an automated composition system, user input is low and the range of output narrow since highly specialized and automated procedures are used to generate the result. Neither kind of system encourages very much user interaction. In a completely automated system, the user only initiates the process and accepts or rejects the output; further modification must be done manually. In a system for computer-realized com-

position, the amount of data required of the user and the typically long calculation times discourage much interaction.

Between these two extremes lie what may be called computer-assisted types of composition, which are characterized by a high degree of user interaction. Such systems attempt to combine both the powerfulness of the automated approach and the open-endedness of the general synthesis system. Specialized procedures are used to speed up the process, but they are designed so as not to restrict the generality of the output. Because such a system does not imply that the user comes with a completely specified composition result in mind, nor that such a result can be completely generated by the system itself, the system needs to furnish the user with the best possible representations of the current compositional state. This is accomplished through data listings, graphics, and, most importantly, through sounding results. Such feedback usually influences, if not actually determines, what the user does next.

A decade ago severe hardware and software restrictions determined how much sound could be delivered to the user of an interactive system in real time or close approximations to it. In the early versions of the author's POD programs (Truax 1973a, 1973b), only a monophonic line of sound could be generated by software in real time. A later polyphonic version of the program became available (Truax 1978), but not in real time; instead, a digital magnetic tape was used to store the calculated sound-pressure function (with a typical 10:1 turnaround time) and play it back later. The advent of hardware synthesizers over the last several years has sparked renewed interest in the question of real-time synthesis and offered some hope to the designers of interactive systems. But because such devices implement a specific synthesis model (or models), they achieve their strength with a corresponding loss of generality. Generality may be lost within the details of the design choices involved in implementing the synthesis algorithm, or more obviously, it may be lost because of the particular choice of algorithm. Therefore, the advent of a completely microprogrammable machine such as the DMX-1000, although not as powerful as any particular hardware synthesizer, is a unique answer to the problem of how to optimize an interactive system of composition without losing the generality of possible output.

However, there are both advantages and disadvantages to an interactive system, such as the following:

Interactive systems must be custom-designed for a particular hardware configuration and optimized for specific musical tasks; such software takes time to develop and is not easy to adapt to another system.

Interactive systems benefit the novice user by providing a readily accessible environment with good learning potential, but such accessibility requires a great deal of documentation and added program strategies so that mistakes are not overly penalized.

An interactive system best serves its users when it can expand to accommodate their musical needs (which frequently increase and change direction as specialized strategies suggest new possibilities), but such expansion requires a resident programmer to implement the necessary changes.

Interactive systems work best when the user has frequent ("on demand") access to sound synthesis; only a dedicated single-user system guarantees such access, and a multiple-user-based interactive system may have technical problems providing its full synthesis capability to all system users.

The PODX system has been designed to operate within a pedagogical as well as a production environment, namely the contemporary music program in the Centre for the Arts at Simon Fraser University (Truax 1982b), where a laboratory instructor is available to work with users. Moreover, it runs on a DEC LSI-11/23 under the RT-11 operating system in a single-user environment. It is primarily designed for studio compositional work, and can be used in conjunction with the facilities of the Electronic Music Studio or totally independently as required. Finally, it is designed to operate with a significant research component in the sense of encouraging advanced students to develop new algorithms and design other system developments. Although

no programmer is on staff as such, the author has functioned in that capacity to develop all of the software described here.

Types of Interaction

Three interaction methods that the user encounters in working with various programs in the PODX system can be identified as follows:

1. Typing on the terminal's keyboard activates specific data to be written directly into the memory of the DMX-1000 while it is running a particular algorithm.
2. The user starts and stops a host computer program to play back a score; the program interacts with the DMX-1000 by scheduling and initiating events, by updating envelope breakpoints, and by interrupting currently sounding events to make room for new events to start when all available voices are in use. In this mode, the program also reports event starts and the current time back to the user, but allows no user control of the performance.
3. One or more scores are performed, as in (2), but the user has control of performance variables via terminal keyboard input, such as starting and stopping scores, changing the speed of performance, and transposing octaves.

All three interactive modes are made possible because the DMX-1000 allows the host computer to update either the DMX-1000 data memory or its program memory between program cycles, which normally means once each sample period. When such updating is under direct user control, as in the first mode outlined above, the effect is similar to the type of interaction found in an analog studio, namely that one hears the sound change immediately in response to a given action. In other words, the sound does not stop while the new input is calculated. This type of interaction works best with the testing of continuous sounds, or what might be called *oscillator mode*. A parameter such as frequency or amplitude can be adjusted while the oscillator continues to function. A typical input format for this kind of control might be a line of data such as the following:

Inc	Frequency	Amplitude	Ramp Inc
1	100_	256	1

The cursor (shown here as an underline) points to one number, and the position of the cursor can be moved to the left or right by the left or right arrow keys respectively. The first number is the size of the increment, and when the up or down arrow key is depressed, this increment is added or subtracted respectively to the number currently pointed to by the cursor. The keyboard controller program multiplies this number by an appropriate scaler and writes the result directly into the DMX-1000 data memory where it is used in the next program cycle to affect the sound. Since the arrow keys produce a repetitive command when held down, a continuous knob-type control can be obtained, with the additional variable that if the increment step is larger than one, a rapid series of discrete steps can be heard in the parameter. In the above example, the ramp increment indicates the size of amplitude step in an attack or decay of the oscillator (e.g., a small ramp increment produces a slow attack). A program convention such as moving the cursor to the far left or right to reinitialize an attack or produce a decay can be included.

For DMX-1000 instruments with multiple voices, a matrix of values, such as those in Table 1, can be placed on the screen. The same kind of keyboard control of each parameter can be achieved with the addition of a convention that some key, such as "line feed," is used to move the cursor to another line of the matrix, that is, to another oscillator. Table 1 gives the example of a thirteen-voice additive synthesis instrument where the user has indicated the frequency of the fundamental and the spread of the harmonics. In this case the spread is one, namely each harmonic is represented (e.g., a spread of two would produce all the odd harmonics). The amplitude, frequency, and ramp increment of each component are separately adjustable; the attack and decay of each component can be ini-

Table 1. Data displayed on the screen for a thirteen-voice additive synthesis instrument. The cursor (_) points to the current parameter being modified.

Inc	*Frequency*	*Amplitude*	*Ramp Inc*
1	1300	1	1
1	1200	3	2
1	1100	5	1
1	1000	2	1
1	900	0	0
1	800	10	2
1	700	7	3
1	600	12	4
1	500	20	6
1	400	25	8
1	300	40	7
1	200	45	9
1	100_	60	10

tialized with the cursor control already described. The attack and decay of the entire group of oscillators can be effectuated with specially designated keys such as S(tart) and E(nd).

These examples are taken from the WAVEX program, which is mainly concerned with creating and testing waveforms stored in a user waveform library. A variety of synthesis algorithms using these waveforms is available within the program, and each has the kind of single or multiple voice testing just outlined. The value of such testing is mainly pedagogical, although the composer may also test out some basic acoustic properties of material that will be worked with later in the format of the stored sound object.

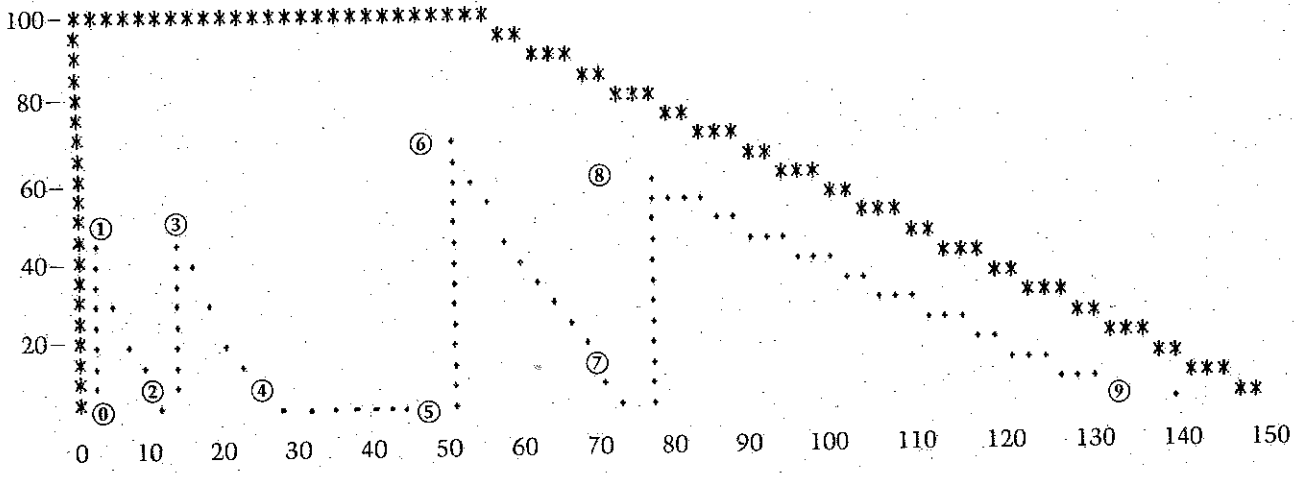
In the PODX system, a sound object is a set of synthesis parameters that describe the envelope(s) and timbral characteristics of a sound within a given model such as FM (see Buxton 1978). Although the sound may be tested at various frequencies and maximum amplitudes, those two parameters are not stored with the object. Rather, the object is later assigned to an event in a score that has an amplitude and frequency, as well as a specific start time and duration. The mode of interaction involved in the testing of sound objects is of the second type listed above, where instead of a score being performed, it is only a single sound that is repeated (with a half-second silence between sounds) until interrupted by the user. The user specifies the sound-object parameters, hears it, changes a parameter, rehears it, and so on, with the option of saving any given version of it for later compositional use. An alternative test mode allows the user to play the object in either monophonic or polyphonic mode from the keys of the terminal, each key producing a different pitch and others, such as the integers 1 to 9, causing octave transpositions.

The program that controls the DMX-1000 for this operation uses a typical clock-interrupt-driven approach. For instance, with the FM sound object, the clock interrupt is set at 1 msec. The carrier and modulating frequencies and waveforms are preloaded into the DMX-1000 and the envelope data for both the carrier and modulating components are precalculated in terms of individual ramp segments. The modulation index envelope is a linear ramp with a variable start value, and when the elapsed number of interrupts equals the duration of the ramp segment in milliseconds, the program initiates the next ramp segment. Intermediate positions along the ramp are calculated by the DMX-1000 algorithm. The amplitude envelope for the carrier is specified in centiseconds, and therefore on every tenth interrupt the program checks whether a new segment of that envelope is to be initiated. This envelope is logarithmic according to a table lookup of logarithmic amplitude factors, and a double-precision ramp value is used for accuracy. Arbitrary waveforms can be assigned to both the carrier and modulator.

From the user's point of view, a sound object is displayed as a set of data on the screen, including a visual representation of the two envelopes, as in Fig. 1. The modulation index envelope (indicated by dots) may be entered in numerical form, or else with graphic input. The latter kind of specification, within the capabilities of the ASCII character screen, involves using the arrow keys to move the cursor around the screen on which the amplitude envelope is displayed, along with a horizontal time line and a vertical percentage scale. Envelope points are plotted by hitting the X key, and envelope input is terminated by hitting the E key. The stored envelope can be displayed and edited in a similar manner.

Fig. 1. Graphic representation of FM object with amplitude envelope (*) and modulation index envelope (.). The vertical scale is percentage of the maximum modulation index, and the horizontal scale is in centiseconds.

Filename: ASC815



Score Performance

The second interactive mode listed above (playing a score) is not interactive from the user's point of view in the same way as the first, that is, it does not involve the user writing directly to the DMX-1000. Instead, the user-specified score and sound object set are available as a source of data that can be precalculated as a linked list arranged in chronological order according to the event start times. Note that the envelope data for all oscillators need not be chronologically ordered because once each envelope is initiated, it proceeds independently of all others. The format of the linked list might take the form of Table 2.

What is needed to perform the score is a scheduler program that handles the timing of event starts and stops, envelope breakpoints, and possible strategies when all available oscillators are in use. Three program levels are operating in such a scheduler:

The DMX-1000 microcoded multiple-voice instrument reads its current values from the DMX-1000 data memory.

A foreground operation in the host computer is interrupt driven at a given clock rate (e.g., 1 KHz). For each voice, it checks whether the DMX oscillator is on, whether it is off but counting down till the instant when the next event is to start, or off entirely. If the oscillator is sounding, a further check is made as to whether it is time to update any envelope segments. After the last envelope segment is finished, it indicates that the oscillator is off and is available to be rescheduled for a new event.

A background operation in the host computer checks whether any voices in the DMX-1000 instrument are free to be preloaded with data for a new event. When such data is loaded, a countdown time (in centiseconds) is set to allow the foreground operation to initiate it. In addition, the background operation compares the current time to the scheduled start time of the next event, and when the difference becomes zero and the event is still not preloaded (because no voice is available), it initiates a procedure to interrupt an existing event to make way for it. Finally, it checks whether events have been started and increments a counter on the screen, as well as a timer when each second has elapsed.

The DMX-1000 is limited in the number of voices

Table 2. Linked data list for each event in FM score

Base address +		
	0	Start time
	2	Carrier sample increment
	4	Modulator sample increment
	6	Waveform no. offset
	8	Address of next event data
	10–22	Amplitude env. data
	24–?	Modulation index env. data

it can produce by the number of DMX-1000 instructions those voices require compared to the maximum number that can be computed in one sample period (256). An effective scheduler for an interactive program must therefore implement a strategy to deal with cases where the user's score requires more voices than can be produced. For instance, if six voices of monophonic FM can be produced at one time (as in PODX) and the user score requests a seventh to start while the six are all sounding, what can or should be done? One option is to stop the synthesis and report the "overflow," as happens in Music-1000. However, the problem of reducing the required score such that it "fits" will probably divert the user from the compositional problem at hand. Another option would be to delay the seventh event until a voice is free to produce it, but this strategy distorts the time structure of the score and possibly confuses the user who might not know how to interpret the results. The solution imposed by PODX is to interrupt an existing event to make way for the new event, with the following rules, similar to those suggested by Kaplan (1981). The event that is closest to being finished is chosen to be interrupted but only if it is in its decay portion. The rationale is that an event that is past its steady state will be the least noticeable if it is terminated early (particularly if masked by five other sounds), whereas if it is cut during its steady state or attack portion, even if ramped to zero, the effect may be too aurally drastic. If none of the sounding events has reached its decay portion, the scheduler waits until one does before interrupting it, thereby establishing a trade-off between acoustic distortion and distortion of the time structure.

Another justification for the strategy of the scheduler in cutting certain events is that in an interactive composition environment, the sounding output need only be an adequate representation of the structure being worked on, and not necessarily its final and best realization. Small transients produced when sounds are terminated seem acceptable to our users because they do not overly detract from the perception of the basic time structure being performed. In an interactive context, the number of voices that can be simultaneously produced by the synthesizer may be less important than the ability of a controlling program to simulate a much larger number by interrupting sounds and preserving the intended time structure. The role of real-time sound synthesis in an interactive system is to provide adequate feedback to the user that allows further compositional choices to be made, not necessarily to generate the final result immediately. Non-real-time sound pressure calculation can always be relied upon for such results once all of the formative compositional decisions have been made and interaction time is not a factor.

Two further remarks about scheduling strategies with the DMX-1000 can be made. The first is that the program executed by the DMX-1000 can also be modified as easily as the data in its memory, even while a performance is being executed. When changing one instruction causes the program to switch between two types of operations, such as changing a ramp from attack to decay, or switching wavetables, or in stereo mode, from left channel leading to right channel leading, such a change can be easily made by the scheduling program at the start, or even during, the sound. Secondly, to prevent possible collisions between requests to the DMX-1000 from the background operation (preloading data) and the foreground operation (updating envelopes), it is possible, and highly desirable, for the background operation to check whether the foreground operation is within a certain time, say 2 msec, of making such a request. Since preloading has lesser priority, it can wait until the high priority foreground traffic reaches a low point before attempting to carry out its operations.

User-Controlled Performance

A further extension of score performance is possible, as described by mode 3 above (performing several scores), when the background program monitors the keyboard for user commands that alter its operation. Although the user is not writing directly into the DMX-1000's memory, as in the first mode, a similar type of command can change the behavior of the scheduling program. The idea behind our program CONDUC is similar to the CONDUCT program described by Buxton (1980), only simpler. From one to six scores are precalculated as linked data lists. Although the events within each score are fixed, the question of which event is to be performed next depends on how the user initiates each score and modifies its speed. Separate bookkeeping is maintained for the start time (and address) of the next event in each file, and it is the job of the scheduler to scan all such files to ascertain which event starts next.

The user controls the performance by manipulating an array such as that shown in Table 3. Performance of a file is started by incrementing the "start switch" from 0 to 1, and is stopped by returning it to 0. Similarly, the "repeat switch" allows the score to cycle back to the beginning when it reaches the end. For simultaneous starts, a "synch switch" can be set; all files with this switch set can be started and stopped with the S and E keys respectively. Two interpretive variables are included within this model, namely a "speed factor" that is initialized at 256 for ease of program calculation, and an "octave transposition factor," initialized at 5. Changes in the speed factor allow each entry delay to be scaled, shorter or longer, and changes in the octave transposition factor multiply or divide frequencies by 2 or its powers. The program involved essentially adds a simultaneous fourth software layer to the three mentioned above, but the ability of even a microprocessor such as the LSI-11/23 to handle such operations in real time attests to the fact that offloading sound sample calculation to the DMX-1000 allows the host computer, as interface, to interact effectively with the user's commands. Different types of interactive control can be tested and optimized, and these could well include real-time generation of the compositional result as well as its interpretation.

Table 3. Sample performance control data for the CONDUC program

Filename	*Start/Stop*	*Cycle*	*Speed*	*Octave*	*Sync*
TEST1	0	0	256	5	0
TEST1	1	1	256	4	0
TEST2	1	0	218	5	1
TEST3	1	1	256	5	1
TEST4	1	1	256	3	0

The Basic Compositional System

As indicated already, the PODX system is comprised of a set of programs that include both general and specialized compositional strategies. The most general part of the system is a set of programs that handle what are called *merge files* or *POD7 files*. These files contain a control block that includes a header describing its size, the number of events, objects, envelopes, voices and so on, plus a user-specified "file comment." In addition, the control block contains all of the sound objects and spectral envelopes (i.e., modulation index envelopes for FM), up to a maximum of 160 objects and 31 envelopes. Following the control block(s) is the score data; each event is specified by entry delay and duration values in centiseconds, a maximum amplitude (on an arbitrary scale of 1 to 60 representing logarithmic increments), a sound object number, and a frequency. An additional *voice number* tagged to each event is useful either for identification during systematic or conditional editing (see below). The voice number can also be used as an indicator of spatial position in a stereophonic performance. Although this score format was originally designed for FM synthesis, it is general enough to be applied to score other synthesis methods, namely waveshaping and additive synthesis. The basic compositional system involves the following set of programs designed to handle the creation, testing, and performance of merge files:

WAVEX: for creating, displaying, and testing waveforms stored in the user's waveform library; these waveforms may be used as part of the sound object.

POD7X: for creating, interactively testing, and storing sound objects; for performing compositional files monaurally or in binaural stereo; or for testing out a file with an alternate set of sound objects.

PDFILX: for copying, editing, listing, and performing all types of compositional files, or for the creation of "dummy" files with default values or those derived from another user file; editing includes both event-by-event editing and "systematic" editing whereby basic compositional selection procedures (e.g., constant, linear, exponential, sequence, aleatoric, aleatoric selection from a group of values, $1/f$ selection, tendency masks) and operators (e.g., linear or percentage change, random change, inversion, time-dependent percentage change) are available; edited files may be performed at any time.

MERGE: for mixing two files in their score format into a third file; the second file can be offset from the start time of the first by any amount; a choral effect variant can be automatically constructed from any file with user choice of number of voices, range of entry delays, and percentage of frequency variation. Score mixing can be performed iteratively, or else up to 10 files, each with its own offset, can be specified and mixed in one operation.

CONDUC: for performing several scores; as described above, CONDUC is a real-time version of MERGE in which the user performs between one and six files by controlling when they start and stop, by asking them to repeat when finished, and by controlling a speed factor and the octave transposition.

As can be seen from this description, the three levels at which compositional decisions can be made are those of the sound object, the score events, and the combination of scores. Each level can be tested interactively and modified. Sound objects and score events can be specified manually, that is, parameter by parameter, or else can be generated as a variant of an existing object or score, or specified by the application of some systematic selection procedure, ranging from deterministic to stochastic.

A particularly powerful aspect of the composition-by-editing approach in the basic system is the introduction of *conditional editing*. The condition is expressed in terms of a chosen parameter having its values in some range according to the predicates "equal to," "not equal to," "less than or equal to," and "greater than or equal to." Buxton (1981) has described a similar notion, namely context in score editing. One can specify, for instance, that a certain editing operation applies to all events whose duration values are less than or equal to 1 sec, or to frequencies higher than a certain value in hertz. In an interactive system, composers often judge the type of modification that a given structure needs in terms of a systematic specification of the events to which the operation applies. Note that the operator involved (e.g., percentage or random variation) can apply to the same parameter referred to in the conditional, for example, all frequencies in a certain range are to be varied randomly within a certain percentage range. By adding the possibility of multiple conditions (up to five), one can express quite complex properties of a group of events, for example, for frequencies above 200 Hz and less than 400 Hz whose durations are shorter than 0.5 sec, and so on.

Specialized Compositional Programs

The PODX system includes specialized compositional programs that implement specific compositional strategies of a more powerful nature. Whereas the basic set of programs described above allows systematic selection of data and other operations on that data, the specialized programs provide a framework for exploring specific compositional problems. The files generated by the use of such programs can be subsequently used in all of the other, more general ones. At present, three such specialized programs have been developed:

POD6X: Historically, this is the original compositional program (Truax 1977) that utilizes the Poisson distribution for calculating frequency/time points within a tendency mask.

Sound objects are mapped onto those events through various selection methods, and the performance is controlled by various "performance variables" that interpret the score.

PDMSKX: A program developed originally for composing one layer of the author's *Arras* (Truax 1982a). It calculates the entry delays and durations of the score events based on how they fit within a tendency mask. A frequency grid in hertz or percentage frequency change is specified by the user to produce a vertical density of events.

PLOTX: A program for calculating *timbral trajectories* (Truax 1983). The user specifies a two-dimensional trajectory on the display screen and generates scores that allow timbres to be systematically related to that trajectory in a variety of ways.

Since density is a major variable within the POD6X program for determining a stochastic time structure, and simultaneous layers of sounds are organized by the other two programs mentioned, the need for a real-time polyphonic sound generation facility is obvious. The incorporation of the DMX-1000 into these programs (as indicated by the letter X in their names) has given the users added incentive to explore the compositional directions that each facilitates. Although a full polyphonic realization may require non-real-time synthesis calculation, even a limited form of real-time sound (currently six voices of FM in monaural mode and five in binaural stereo) is very useful for the exploratory stages of a composer's work. Having such synthesis available may also encourage more experimentation since it can be done quickly and efficiently. In addition, a new compositional direction may require alternate synthesis procedures, as with PLOTX, which needs linear amplitude envelopes for its realization. The microprogrammability of the DMX-1000 aids such new developments.

The DMX-1000 is also used in the non-real-time synthesis facility (POD7) that mixes its calculated sound pressure functions on disk and transfers them to digital magnetic tape. The completed tape is played back via the direct-memory-access (DMA) interface to the DMX-1000 where the samples are converted with its 16-bit digital-to-analog converters (DACs). The technique of data transfer is a standard multiple buffer approach where samples are read from magtape into memory while a parallel operation transfers them out to the DMX-1000. The limiting speed factor is the speed of the tape (45 ips), and the resulting sampling rate is 32 KHz for monaural mode, and half that per channel of stereo. Since the full calculating potential of the DMX-1000 is not being used in such a transfer, recent program developments have added optional "post-processing" of the stored data, including digital reverberation and filtering.

Real-time digital signal processing is well within the capabilities of the DMX, either with Music-1000 or with custom-designed software with similar interactive potential to that described here. The digital delay and analog-to-digital converter (ADC) units marketed by Digital Music Systems are specifically designed for such work; although this hardware is not included in our present system, we have begun to develop some software in this area using other converters and the LSI-11's own memory for operations with digital delay.

Conclusion

The advent of digital hardware designed for sound synthesis and signal processing is important not only for live performance instruments but also for the development and use of interactive composition systems to provide the aural feedback of compositional results. With suitable programming, various types of interaction with such a device can be implemented where the host computer performs a large amount of data interpretation and generation, as well as complex scheduling operations. With synthesis handled by hardware, the host computer can function as an "intelligent" interface between the user and the synthesizer. Therefore, such devices can be used both to facilitate the compositional process and to investigate new forms of composer-machine interaction.

Although more hardware synthesizers are becoming available, very few at present are microprogrammable like the DMX-1000. For any compositional facility that is intended to be open-ended and/or re-

search-oriented, this characteristic is of the greatest importance. Current synthesis procedures can be refined and custom designed for any particular application, and as new procedures are suggested, they can be easily implemented and integrated within an interactive testing facility. It seems unfortunate that the concept of microprogrammability is not more widely used in computer music system developments since its flexibility is ideally suited to a field that is rapidly expanding.

The PODX system is the author's ongoing and constantly developing series of programs that exploits the interactive potential of the DMX-1000 within the context of interactive studio composition. The software is available free of charge to noncommercial studios. It is written entirely in FORTRAN and Macro 11 Assembler, plus DMX-1000 microcode, and assumes the RT-11 operating systems. One hopes that the availability of a variety of alternative software packages will encourage the musical usefulness of the DMX-1000.

References

Buxton, W. et al. 1978. "The Use of Hierarchy and Instance in a Data Structure for Computer Music." *Computer Music Journal* 2(4): 10–20. Revised and updated version in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press.

Buxton, W. et al. 1980. "A Microprocessor-based Conducting System." *Computer Music Journal* 4(1): 8–21.

Buxton, W. et al. 1981. "Scope in Interactive Score Editors." *Computer Music Journal* 5(3): 50–56.

Chowning, J. 1973. "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation." *Journal of the Audio Engineering Society* 21(7): 526–534. Reprinted in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press.

Kaplan, S. J. 1981. "Developing a Commercial Digital Sound Synthesizer." *Computer Music Journal* 5(3): 62–73.

Koenig, G. M. 1970a. "Project 1." Electronic Music Reports 2. Utrecht: Institute of Sonology.

Koenig, G. M. 1970b. "Project 2—A Program for Musical Composition." Electronic Music Reports 3. Utrecht: Institute of Sonology.

Mathews, M. 1969. The Technology of Computer Music. Cambridge, Massachusetts: MIT Press.

Truax, B. 1973a. "Some Programs for Real-time Computer Synthesis and Composition." *Interface* 2: 159–162.

Truax, B. 1973b. "The Computer Composition—Sound Synthesis Programs POD4, POD5 and POD6." Sonological Reports 2. Utrecht: Institute of Sonology.

Truax, B. 1977. "The POD System of Interactive Composition Programs." *Computer Music Journal* 1(3): 30–39.

Truax, B. 1978. "Computer Music Composition: The Polyphonic POD System." *IEEE Computer* 11(8): 40–50.

Truax, B. 1980. "The Inverse Relation between Generality and Strength in Computer Music Programs." *Interface* 9(1): 49–57.

Truax, B. 1982a. "Timbral Construction in *Arras* as a Stochastic Process." *Computer Music Journal* 6(3): 72–77.

Truax, B. 1982b. "The New Electronic and Computer Music Facility at Simon Fraser University." In *Proceedings of the 1982 International Computer Music Conference*, ed. T. Blum and J. Strawn. San Francisco: Computer Music Association.

Truax, B. 1983. "The Compositional Organization of Timbre in a Binaural Space." Presented at the 1983 International Computer Music Conference, Rochester, New York.

Vercoe, B. 1980. "Reference Manual for the Music 11 Sound Synthesis Language." Cambridge, Massachusetts: M.I.T. Experimental Music Studio.

Wallraff, D. 1979. "The DMX-1000 Signal Processing Computer." *Computer Music Journal* 3(4): 44–49.