MODELS OF INTERACTIVE COMPOSITION WITH THE DMX-1000 DIGITAL SIGNAL PROCESSOR

Barry Truax
Dept. of Communication & Centre for the Arts
Simon Fraser University
Burnaby, B.C., Canada

Recent digital hardware developments, including both digital synthesizers and programmable signal processors such as the DMX-1000 (Wallraff 1979), are sufficiently powerful to allow real-time polyphonic synthesis of sound and/or signal manipulation. Therefore they may be profitably integrated within interactive systems for composition and sound synthesis such as the author's PODX system (Truax 1985) which is based on an LSI 11/23 machine equipped with a DMX-1000. The ability to hear intermediate compositional results in real-time facilitates the learning potential of the system, and encourages both exploratory and optimization modes of composer-machine interaction. This paper presents the specific models of interaction that have been designed and implemented with the current system.

The general concept of the interactive approaches I will describe is that the host machine, such as practically any microprocessor, acts as an intelligent interface between the user and the synthesis device. Because the burden of sound synthesis or processing is carried by the synthesizer, the host processor can devote its capabilities to control tasks, as well as to those initiated by user commands. Further, if the software allows interrupt programming techniques to be used, more than one task may be carried out in parallel. Although the idea of the host processor acting as a controller is hardly new, the extent of its potential role as intelligent interface between the user and a synthesizer is not always given a great deal of thought, although performance oriented systems have tended to develop that potential further. Often the details of the input devices, control languages or the synthesizer design itself are given the most attention. On the other hand, the speed and programming architecture of microprocessors is well suited for designing real-time control methods that allow the user to interact in a meaningful way with the sound producing device.

1. DIRECT DATA MEMORY UPDATES

Probably the simplest form of user-machine interaction involves the user writing directly into the data memory of the synthesizer. A typical instance is shown in Fig. 1. In this model, and the others presented here, the input and output modality for the user is the conventional CRT keyboard and screen. Other devices may equally well be substituted as desired.

The control program maintains a dual array of current data, one of which represents the user's version of the data. Data from this array is updated on the screen when each change is made. The control program also knows how to translate the user's data into that which is appropriate for the synthesizer. Some such translations stem from the synthesis algorithm, for instance when a desired frequency is translated into a sample increment for a wavetable in the synthesizer. Other translations implement a convenient scaling of a parameter for the user so that small changes in a parameter produce a just noticeable difference in the output. For instance, when the user increments an amplitude value, the program can increment the amplitude scaling factor used by the synthesizer by a larger amount to create a perceptible increase in loudness.

The actual control method used in the PODX system involves a line or array of integers that appear on the screen as follows:

INC	FREQUENCY	AMPLITUDE	RAMP INC
1	100	256	1

The user uses the left and right arrow keys to position the cursor at the variable to be controlled. The up and down arrow keys add or subtract the INC value to the current parameter; such values may be constrained by upper and lower limits. The INC value itself may be incremented or decremented so that parameters may be stepped by values other than 1. Most CRT

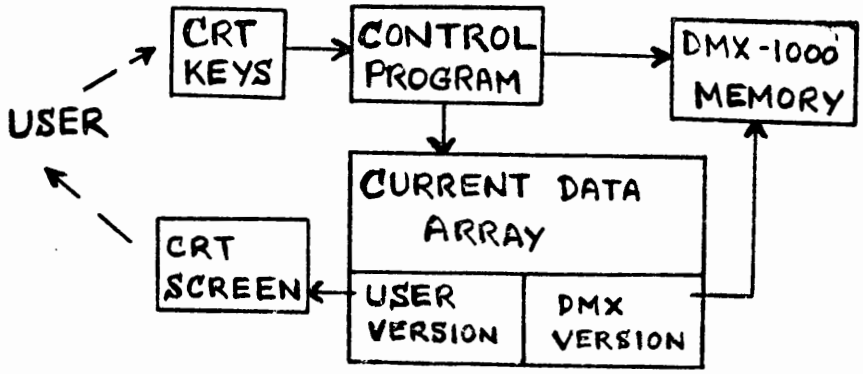


Fig. 1 Interaction Mode for Direct Data Memory Updates

keyboards allow for a repeated command when a key is held down, in which case a quasi-continuous variation of a parameter may be effected. Additional commands may involve pushing the cursor to the far right or left, for instance to initiate an attack or decay. Other keys may be coded for specific operations as well. Similar control could come from a mouse or other input device.

The above example implies a single oscillator in test mode, or an oscillator bank if the line of integers is replaced by a two-dimensional array where each line refers to a specific voice. However, the same type of control could be applied to a signal processing task. The PODX system contains the example:

BLOCK	SPEED	FILTER CUTOFF	DELAY
1	152	5000	100

As each block of sound is played from the disk or tape, it can be altered in speed by one variable, filtered (high-pass or low-pass) with a given cutoff frequency, and processed with delay networks (e.g. comb filter, reverberation or chorus effect). All parameters under user control are translated into updates to the data memory of the DMX-1000 to alter its current program.

It should be emphasized that all control changes in this mode are immediately heard without the sound being stopped, similar to operations in an analog studio. Therefore, the user has the advantage of precise digital control combined with the interactive feel of analog processes.

2. USER INTERACTION WITH PRE-CALCULATED DATA

A greater degree of performance capability can be achieved when data that has been pre-defined by the user is available. The simplest form might be a single sound object, or a set of such objects each having its own timbral characteristic. More conventionally, the available data might take the form of a score. In any case, a scheduling program is required to control the performance of the data, as shown in Fig. 2.

The scheduler will probably need to include two task levels, a foreground level for precisely timed actions, and a background level for tasks that may be performed at lower priority levels. A programmable clock can be set to generate interrupts at a given rate (1 ms in the PODX programs). The background operation has access to pre-compiled data lists which have been translated from the user defined score or sound objects. The purpose of pre-compiling this data prior to run time is simply to save calculation time during the performance. The background operation is mainly concerned with pre-loading event data into the DMX-1000 and preparing the way for the foreground operation to execute its tasks that are time specific, such as initiating events and updating envelopes. However, the background operation also has time to accept commands from the user via the CRT keys and to display current information back on the screen.

The current types of CRT key control used by the schedulers in the PODX system depend on the kind of data that the user is performing. In the case of the single sound object, the sound is initiated whenever the user hits a key. Individual keys cause the next event to be heard at a different pitch, thus allowing the object to be tested at different frequencies, or to be heard in counterpoint or textures. A simple pitch control has been implemented whereby the integers from 1 to 9 transpose the object to different octave ranges, and the letters A through Z transpose the event within the octave. Other types of frequency grids or tuning systems could also be constructed.

In another test mode where a set of different objects is available, the

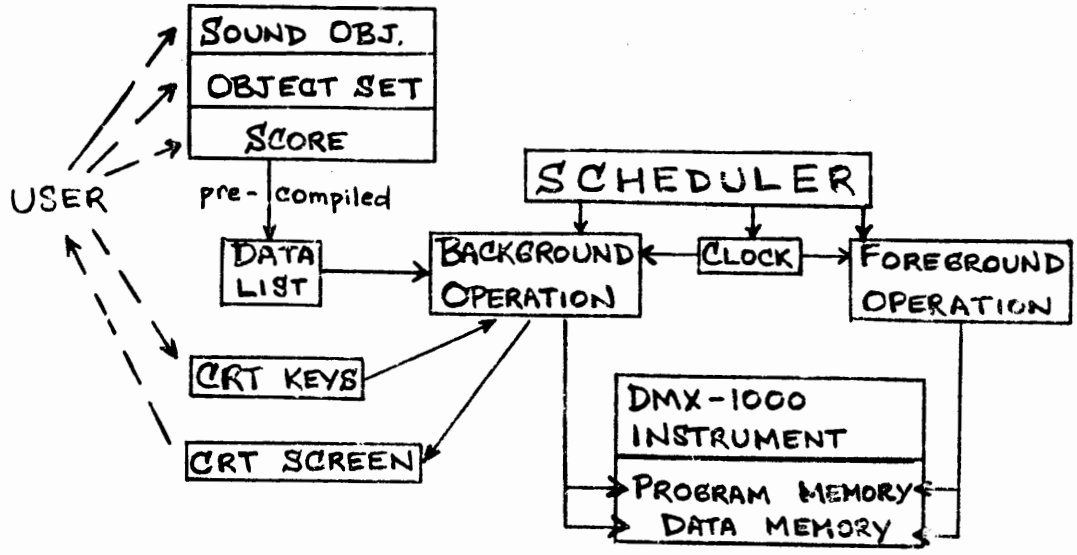


Fig. 2 Interaction Mode with Pre-calculated User Data

integers 1 through 9, followed by 0, followed by A through Z, specify which object in the set is to be heard. Transposition is effected by the control method referred to earlier where the cursor points to the frequency parameter which can be incremented with the up arrow key or decremented with the down arrow key.

In the case of a pre-defined user score, the user starts and stops the performance from the keyboard, although more extensive control in such a situation will be described in the next section.

A typical example of what can be involved in the foreground and background operations referred to here is presented in simplified form in Fig. 3. With the DMX-1000 instruments used in the PODX system, envelope ramps are generated by the DMX-1000 itself, and breakpoints are serviced by the host computer. Therefore, the main task of the foreground operation is to update envelopes at their breakpoints by giving the new ramp increment value, duration, etc. of the next envelope segment. Precise timing is required, hence the need for clock control. In addition, when a score is being performed, the event must be initiated at the precise time specified in the score.

For each voice, the foreground operation first looks at a variable which tells it whether that voice is off, in which case nothing more needs to be done, whether it is on, in which case the program checks if it is time to update an envelope, or whether it is pre-loaded and waiting to be started. In the latter case, the variable indicating how many time units must elapse until the event starts is decremented, and when it reaches zero, the event is initiated and envelope data is loaded. Event starts and terminations include setting variables which are then available to the background operation for its use, such as indicating to the user the number of the event that has just begun.

The background operation pre-loads event data, either on the keyboard command of the user or else according to a score. Similar to a taxi dispatcher who must match incoming calls with available cars, the background operation assigns available synthesizer voices to events that must be generated. It checks which voices are free and when it finds one, pre-loads all data other than the envelope data and sets the countdown variable for the foreground operation to use in its timing. If the event is to start immediately, the countdown variable is set to 1.

In the case of a score, the current time must be compared to the start time of the next event, and when these times become equal and the event is still not loaded because all voices are busy, a strategy for interrupting a sounding event must be implemented. The PODX system uses the strategy of finding the event that is closest to being finished, as long as it is in the decay portion of its envelope. Early termination of that event is probably the least objectionable to the user. Switches for whether the next second has elapsed and whether any events have been initiated are periodically checked and the appropriate information on the CRT screen is updated for the user's benefit. Finally, once an event is loaded, the program checks whether the score is done, and if so, whether all events have finished sounding. In the case of sound object performance, the background operation takes information about which key has been struck in order to load and

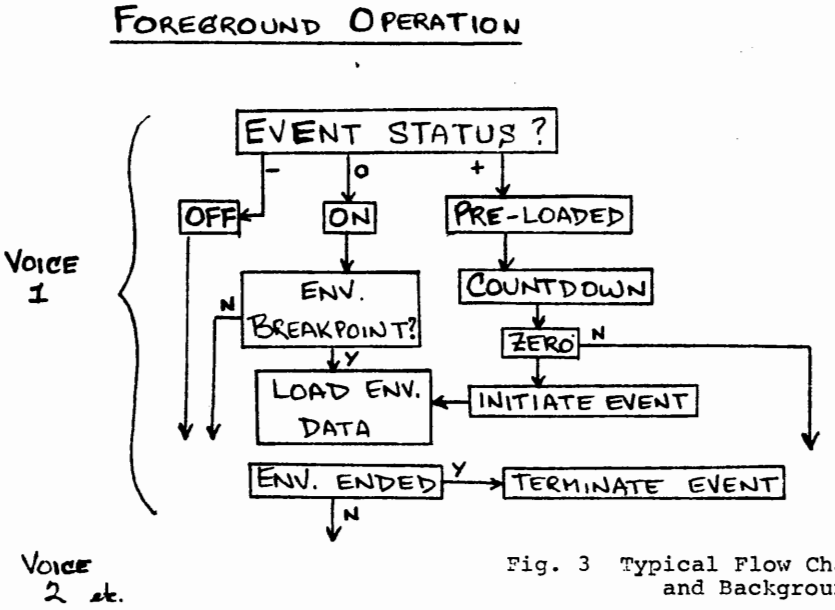
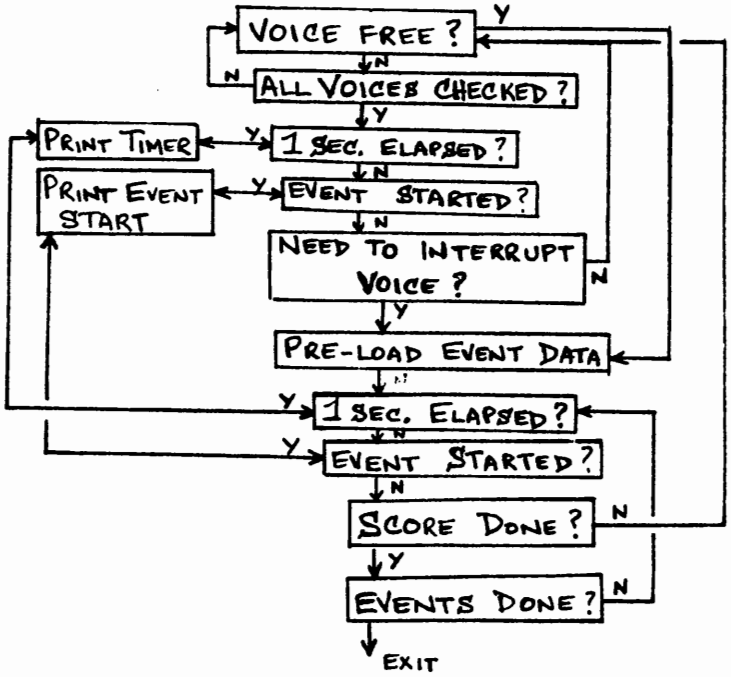


Fig. 3 Typical Flow Chart for Foreground and Background Operations



execute an event, as well as to select which object is to be performed at what frequency, as already described.

One final note of precaution about scheduling in this manner is that the background pre-loading of data into the synthesizer must not collide with high priority data loading from the foreground operation. Essentially each program level cannot try to communicate with the signal processor at the same time. However, since all activities of the foreground level depend on counters telling it when to initiate certain actions, these counters can be checked by the background operation to see whether any are within 1 or 2 counts of initiating an action. Since the activities of the background operation are of lower priority, they can be postponed until there is no imminent foreground activity.

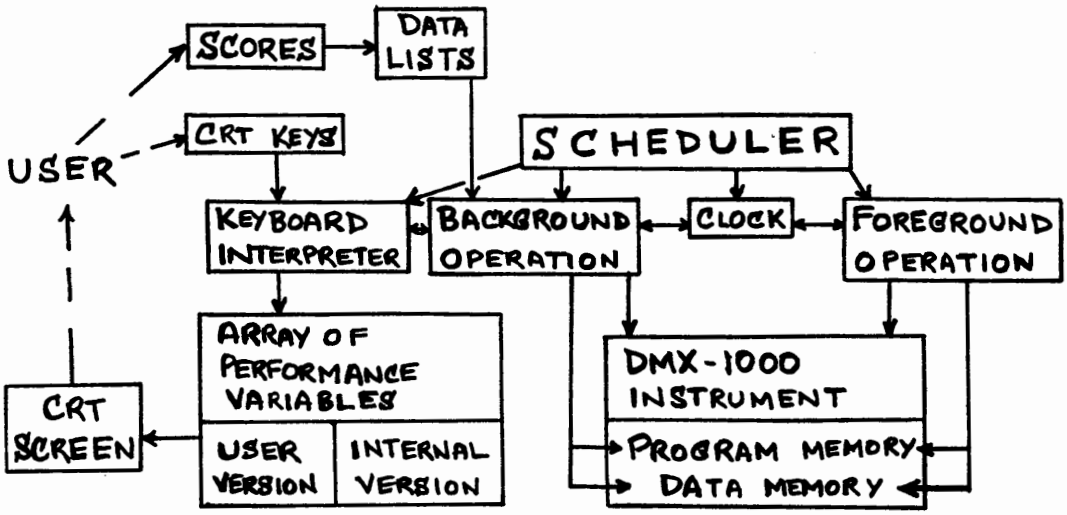


Fig. 4 Interaction Mode for User Controlled Score Performance

3. USER CONTROLLED SCORE PERFORMANCE

The final interactive mode that will be presented is in some ways a combination of the other two, as shown in Fig. 4. Scores are scheduled in the manner already described, but their performance is controlled by the user manipulating a set of performance variables on the CRT screen similar to the control method introduced in the first model. Also, similar to the first model, the performance variables are stored in two versions, one that is presented to the user, and the other that is a set of internal data that may be used by the background operation. This type of interaction requires a third program level, which may be called the keyboard interpreter, in addition to the foreground and background operations.

The program that uses this type of control is similar to the CONDUCT program described by Buxton (1980), only simpler. A typical array of control data might look as follows:

SCORE	START/STOP	CYCLE	SPEED	OCTAVE	SYNC
TEST1	1	1	256	4	0
TEST1	1	1	192	5	0
TEST2	0	1	256	5	1
TEST3	0	1	256	5	1

In the PODX version, up to six files may be performed by the user, each being initiated when the user toggles the START/STOP switch from 0 to 1. The file can be repeated by setting the CYCLE switch, its speed altered by changing a speed factor, its frequencies transposed by changing the OCTAVE value, and so on. In order to be able to start two or more files simultaneously, a SYNC switch can be set, and when a particular key is hit the files thus indicated can be started or terminated.

Although each score has its own specific internal time structure, it is not known which event from which score should be performed next until the user initiates the performance. Moreover, since the speed of the performance of each file can be separately controlled, the scheduling program must be capable of fairly elaborate bookkeeping to keep everything straight. It is illustrative of the point I am making that a microprocessor such as the LSI 11/23 has no trouble keeping up with such demands, even while updating 12 or more envelopes. If such is the case, then it is also clear that more elaborate and interesting forms of user interaction with scores and related sound data are possible if only one designs the appropriate software.

4. CONCLUSION

The key to understanding and designing such control systems is recognizing their inherent parallelism and use of hierarchy. Feedback of information between levels is also an important component. As such, these programs seem to resemble organic systems more than sequential machines. Debugging such programs requires a different approach too as their complete behaviour can only be observed when running in real-time. When stopped or single-stepped, these programs lose their most essential quality - the time dependence of the various levels operating in parallel. The user's actions merely form another level of data flow that changes the behaviour of the other levels of operation.

The examples I have presented here are only a sample of some of the more obvious kinds of interaction that the host processor is able to support between the user, pre-defined data, and a synthesis device. An important variant that has not been mentioned (or yet implemented in the PODX system) is that where the score data is not pre-defined, but rather composed and modified in real-time. Joel Chadabe (1984) and Martin Bartlett (1979), among others, have used this approach extensively; Chadabe refers to it as "process automation" and "interactive composing." Live performance oriented systems often use a control structure along the lines suggested here, particularly when they must react to live performers or respond to the gestures and instructions of a composer-performer.

In fact, one of the conclusions I reached in a recent survey of the impact of electroacoustic technology on communicational processes including music (Truax, 1984), was that future developments in computer systems may bridge the traditional gap between the studio composer who does not compose in real-time and the improvising performer who does. The required rates of data flow may be met by future hardware developments, but what will also be required is a practical implementation of the knowledge base that allows the composer to structure, shape and control sound material. The work reported here seems to me to have characteristics that suggest the path toward such an intelligent interface between the composer and sound processors.

REFERENCES

Bartlett, M. (1979). Microcomputer-controlled synthesis systems for live-performance. Computer Music Journal 3(1), 25-29.

Buxton, W. et al. (1980). A microprocessor-based conducting system. Computer Music Journal 4(1), 8-21.

Chadabe, J. (1984). Interactive Composing: An overview. Computer Music Journal 8(1), 22-27.

Truax, B. (1984). Acoustic Communication. Norwood, N.J.: Ablex.

Truax, B. (1985). The PODX system: Interactive compositional software for the DMX-1000. Computer Music Journal 9(1).

Wallraff, D. (1979). The DMX-1000 signal processing computer. Computer Music Journal 3(4), 44-49.