



**CONSIGLIO NAZIONALE
DELLE RICERCHE**

Andrea Ballista ^, Enea Casali ^,
Jacques Chareyron ^, Goffredo Haus ^

**A MIDI/DSP Sound Processing Environment
for a Computer Music Workstation**

N. 7/113

11/92

**PROGETTO FINALIZZATO
SISTEMI INFORMATICI E CALCOLO PARALLELO**

SOTTOPROGETTO 7

SISTEMI DI SUPPORTO AL LAVORO INTELLETTUALE

**PROGETTO FINALIZZATO
SISTEMI INFORMATICI E CALCOLO PARALLELO**

SOTTOPROGETTO 7
Sistemi di Supporto al Lavoro Intellettuale
Coordinatore Cesare Maioli

Andrea Ballista [^], Enea Casali [^],
Jacques Chareyron [^], Goffredo Haus [^]

**A MIDI/DSP Sound Processing Environment
for a Computer Music Workstation**

N. 7/113

11/92

Reprint

[^] LIM - Laboratorio di Informatica Musicale
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano.

Andrea Ballista, Enea Casali, Jacques Chareyron, and Goffredo Haus

L.I.M.—Laboratorio di Informatica Musicale
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
via Comelico, 39
I-20135 Milan, Italy
Music@imiucca.csi.unimi.it

A MIDI/DSP Sound Processing Environment for a Computer Music Workstation

This article describes the Sound Processing Environment (SPE) of the Intelligent Music Workstation (IMW). IMW-SPE integrates MIDI devices and digital signal processing (DSP) units, allowing the musician to control both within the same sequencing environment, so that DSP units may be considered as if they were real-time MIDI devices with timbre control capabilities. IMW-SPE consists of three subsystems; the MIDI/DSP sequencing program, and two timbre programming software modules (Waver and LASy Workbench) that implement a number of digital sound synthesis techniques via DSP microprogram units. The end-user has complete control of the whole environment by means of MIDI programming without the need of DSP microprogramming.

Background

The last ten years have seen a dramatic growth of two parallel lines of development within computer music applications, one based on MIDI devices (synthesizers, samplers, mixers, effects, synchronizers, sequencers) and one based on DSP units (add-on boards, standalone units, matrix-connected DSP systems).

MIDI devices have become popular owing to features such as high reliability, ease of use, low cost, and related software availability, while DSP units have not, because of the difficulty of programming them, the lack of ready-to-use software, higher prices, and sometimes even poor reliability.

On the other hand, MIDI devices have intrinsic limitations, particularly with respect to dynamic

timbre control and special temperament (Loy 1985), while DSP units allow more flexibility, and easier control of time-varying spectra, and cost increasingly less. Furthermore, the growing computing power of personal computers now allows the control of DSP units with general-purpose personal computers for real-time applications.

We thought that a good solution would be to combine the positive features of MIDI devices and DSP units. In particular, the flexibility of DSP units and the ease of describing musical event streams with MIDI seem to us the most interesting features to combine; in fact, MIDI codes are commonly accepted, well known, and widely used, while DSP units allow complete control of sound processing.

We have implemented our system by allowing the control of both MIDI devices and DSP units within the same sequencing environment, so that DSP units may be considered as if they were real-time MIDI devices with sound processing capabilities. Our work was developed within the framework of the Intelligent Music Workstation (IMW) Project, a three-year project (June 1989–December 1992) devoted to the design and development of a music software and hardware environment in which commercially available products are integrated together with prototype modules built within the framework of a music information research project (Camurri et al. 1989). Two research groups are involved in the IMW Project: the L.I.M. at the University of Milan, and the Dipartimento di Informatica Sistemistica e Telematica (DIST) at the University of Genoa.

Most of the prototype modules run on the Apple Macintosh family of computers. All the software modules are able to import and export MIDI files in the different formats available. Soundfiles are compatible with the Digidesign Sound Designer II for-

mat. A hypertext musician-workstation interface is provided to enhance operative simplicity.

The IMW is an open environment, so that musicians can add their own applications without any trouble due to special music data formats at both the MIDI and sound levels of representation.

Among the many software modules of the IMW, we have developed a specific sound processing environment in which both MIDI devices and DSP units are integrated and both sequencing and timbre programming functionality are implemented.

The Sound Processing Environment

The software architecture of the sound processing environment we have built allows the musician to control both MIDI and DSP units; it is based on an Apple Macintosh computer equipped with some number of Digidesign Sound Accelerator DSP cards (Stiglitz 1991). Audio signals coming out of these cards and MIDI devices are mixed with an audio mixer. The main question was, how we could create a set of integrated applications for synthesizing new sounds with the DSP cards and concurrently control both external MIDI devices and internal DSP cards.

The first step was to think of the DSP subsystem as a standard synthesizer with a MIDI interface. By loading different microprograms onto a DSP card, the musician can choose among different *virtual MIDI synthesizers*; in this way one can have more than one instrument in a single piece of hardware because the DSP is much more flexible than a common synthesizer, as we will show below. This point of view was necessary to get a software application that controls both MIDI and DSP units at the same time; the DSP-Synth idea also proved useful for creating new sounds using DSP cards.

We developed a sequencer (SQAN) for MIDI/DSP handling and integrated performance execution, and some DSP-based applications (LASy Workbench, Waver) for timbre building by means of well-known digital sound synthesis techniques.

To allow a set of integrated software tools for information exchange, modules for reading and writing standard data interchange formats are needed. The

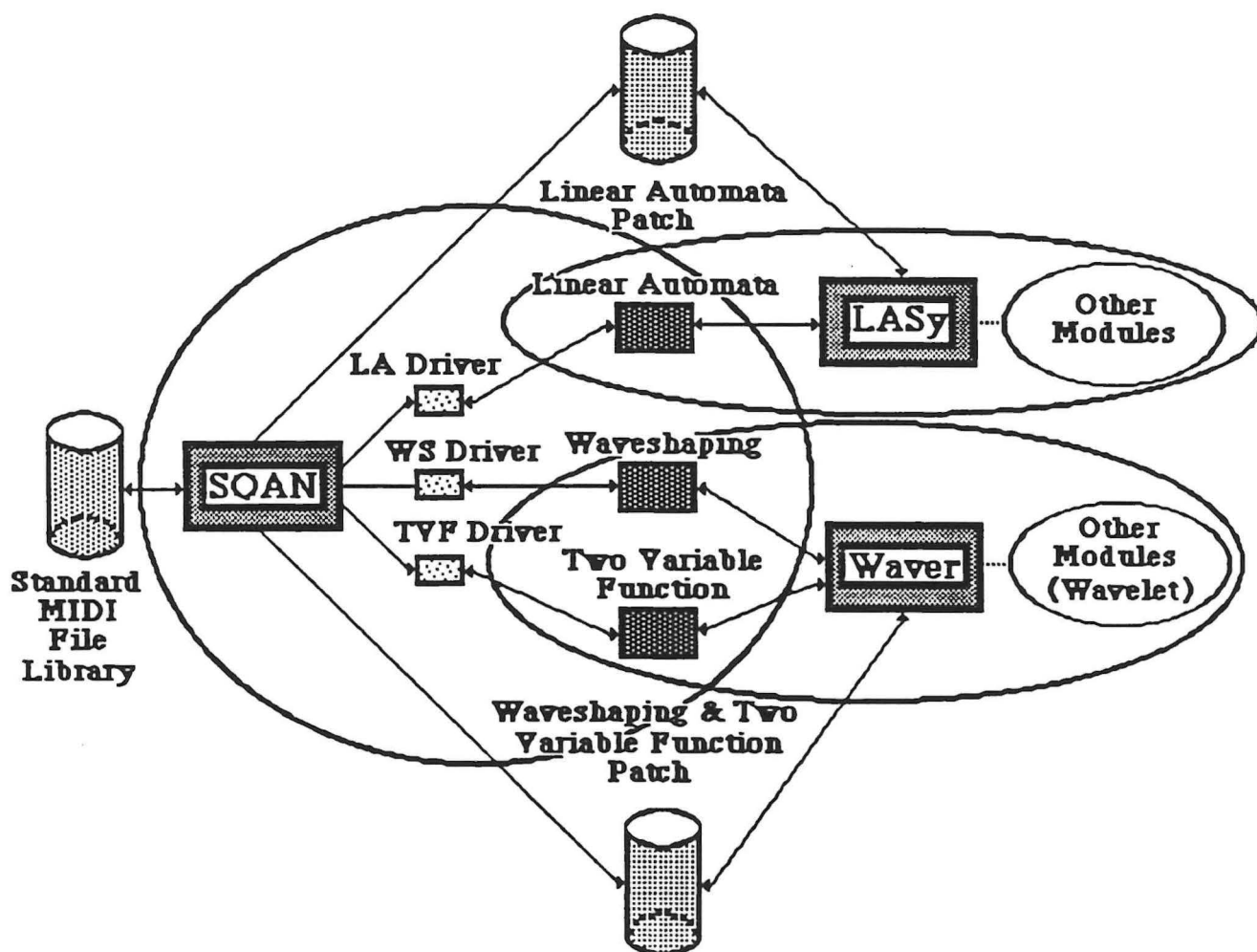
IMW-SPE uses three of these standards: IMA Standard MIDI File 1.0 and Apple MIDI Management Tools 2.0 for MIDI data exchange, and Digidesign Sound Designer II soundfile format for audio data exchange. We have also defined a private file format for exchanging sound synthesis parameters between SPE applications, the Patch File format.

The two stand-alone applications for sound creation and experimentation are shown in Fig. 1: LASy Workbench (for Linear Automata Synthesis) and Waver (for two-variable function synthesis and waveshaping). SQAN is the MIDI/DSP sequencing application, which includes slightly modified versions of routines and microprograms taken from Waver and the LASy Workbench. This integration process will be explained below in the section on MIDI/DSP sequencing. SQAN is fully compatible with the Apple MIDI management tools; it can exchange standard MIDI files with any application or device that supports them and uses patch files to communicate with the LASy Workbench and Waver. Every Patch File has an identification code that specifies the synthesis technique used, so that both LASy Workbench and Waver can read and write Patch Files with their specific DSP settings. In this way SQAN can exchange patch files with both of the applications for real-time timbre control.

Waver and LASy Workbench have other special features (not available in the SQAN application). They also have their own (larger) set of microprograms for digital sound synthesis; the LASy Workbench has some linear automata synthesis algorithms better suited for sound experimentation, while Waver lets the musician experiment with the wavelet transform synthesis, which is not implemented in SQAN due to the large amount of data and computation time required.

It should be stressed that the Sound Processing Environment is an open system. Any programmer can write another DSP module for sound synthesis using the functions provided in the system. To build a fully working version of SQAN that integrates a new synthesis technique, the sequencer shell must be linked with a dedicated DSP driver, patch file functions for data exchange, and functions that take care of the user interface and the Motorola DSP56001 microprogram.

Fig.1. Software architecture of the SPE.



-  High level Macintosh code (Pascal, C)
-  Low level Macintosh/DSP code (Assembly Motorola 68000 and 56001)
-  DSP code (Assembly Motorola 56001)

MIDI/DSP Sequencing

SOAN is a basic sequencing application that lets the musician record a timed MIDI sequence and send it to a set of MIDI devices and to some number of DSP cards. SOAN is able to handle up to 32 tracks of MIDI data that may be sent to a MIDI device or a

DSP card or both, according to the setting of a software flag. The MIDI/DSP sequencer can read and write MIDI data to and from files, and is fully integrated with the Macintosh MIDI management tools so that it can exchange MIDI data with other applications, according to the connections made from the MIDI management tools patch bay panel. This se-

quencer also includes a mixing feature for the key velocity control of MIDI note-on and note-off events of any of the 32 MIDI tracks; the musician can see a 32-track mixing console on-screen, and each track may be controlled by a fader while the sequence is running. We did not try to provide SQAN with all the features that can be found in commercially available sequencers, such as complete editing functions. If the musician needs it, he may load sequences to and from his favorite sequencer program using the standard MIDI files. SQAN's most relevant feature is the MIDI/DSP link, which allows the full integration of DSP power with the MIDI world.

MIDI/DSP Integration

When the sequencer is in "PLAY" mode, it reads a MIDI data sequence. If a track has been assigned to a DSP card (by setting the relevant flag), the sequencer has to call a driver (particular driver to each synthesis technique), which translates the MIDI messages into the appropriate code to send the DSP card. Therefore, the first thing we have to do to integrate the MIDI and DSP worlds is to define the relations between standard MIDI commands and DSP actions. Not all MIDI messages are interesting for a first approach to MIDI/DSP integration; in the current implementation the only MIDI messages passed to the DSP are *note-on*, *note-off*, and *program-change*. For the first two messages, the translation is obvious:

note-on → sound begin operations; and
note-off → sound end operations.

When SQAN passes a note-on message to the driver, the driver has to send the right commands to the DSP card so that sound can be generated at a specific frequency (key number) and dynamic (key velocity). How the chosen frequency will be computed is strictly dependent on the sound synthesis technique being used. The key velocity parameter can be freely interpreted; it may cause changes in the envelope parameters of the original sound (amplitude, attack, etc.), or may act directly on the timbre characteristics of the sound.

The note-off message stops the note of a specific frequency (key number) that is currently active on

the DSP card. The driver decides how it wants to interpret this command. The crudest form of translation is the setting of the card volume to 0, but a less drastic driver will prefer to switch to the release segment of the envelope when it receives the note-off message. Naturally, it is possible to use the key velocity parameter to control the release rate.

The driver is responsible for the routing of notes that it sends to the (software and hardware) sound generators according to the density of polyphony of the current synthesis module. Particularly, it must decide about which actions have to be taken when it receives a note-on message and no more synthesizer voices are available.

The translation of the program-change command is less straightforward. When a common MIDI device receives such a message, it changes the parameters of the sound synthesis and the result is a modification of the timbre of the generated sound. The sets of parameters corresponding to every possible value of the program-change number are stored in the sound module memory (whether by the factory programmers or by the instrument user), and are generally named *patches*. We have adopted the patch name and concept for timbre control within SQAN, and more generally the SPE.

A patch is a set of parameters that define the timbre of the sound generated by a synthesis module implemented in SQAN. The number and meaning of the parameters in a patch depend on the chosen synthesis method, and these parameters may have reference tables that are specific for each synthesis module. When we want to change the timbre of the current DSP sound we have to send a MIDI program-change message to the SQAN sequencer, which will route it to the DSP driver.

This implementation is consistent with the everyday practice of MIDI users. Standard MIDI files with program-change commands imported in SQAN will behave in a similar way whether a track is directed to an external sound device or to the internal DSP generator. On the other hand, this choice limits to 128 the number of elements in a patch file and is not as flexible as it would have been had we resorted to the implementation of timbre control by means of MIDI system exclusive messages.

How does the SQAN user interact with patches

and synthesis modules? The *DSPControl* menu lets him or her choose the DSP synthesis module he or she wants to use. Another way to control the synthesis is to choose "Open Patch File" from the "File" menu, causing the synthesis initialization specified in the file. After choosing a synthesis module, we may open a window to see which patch file is opened, to modify existing sounds, and to build new ones. Each sound inside a patch file has an identification number (0 through 127) that corresponds to a program-change number. Another edit window handled by SQAN offers complete control of the flow of program-changes that the musician can browse by identifier. In this way the musician has two main working environments: the DSP editing environment (a specific one for each synthesis module) that is handled by the synthesis module itself; and the MIDI program-change edit environment that is handled by SQAN.

It is important to note that a program-change command may be sent to the DSP card at any time. The timbre-modifying request will be processed in real time, according to the synthesis technique, even if there are some note-on messages pending. Because of this, every kind of sound manipulation is theoretically available in real time by means of MIDI program-change messages.

Software Modules and DSP Drivers

We have introduced SQAN as an open application and said that it may include a number of microprograms that implement various kind of synthesis techniques. To allow a smooth integration process, we have defined a set of standard functions that microprogram developers have to write and another set of functions that they may use. Figure 2 shows the two main software subsystems, SW-Sequencer and SW-Synthesis, that contain the following functions used in the integration mechanism, where each synthesis module has to handle its own set of submodules.

INIT Synth Module—this submodule includes the handling of the initial operations to be performed at the launching of the SQAN applica-

tion; it is called once by SW-Sequencer at launching time.

DISPOSE Synth Module—this submodule includes the operations to be performed on SQAN application shut down; it is called once by SW-Sequencer on shut down.

OPEN Synth Module—this submodule includes the operations to be performed every time the synthesis module is opened; note the double-ended arrow between this module and the SQAN Operative Shell (SOS), which means that the opening operation may be requested by either SW-Sequencer or SW-Synthesis.

CLOSE Synth Module—this submodule includes the operations to perform every time a synthesis module is closed; as in the OPEN case, note the presence of double-ended arrow.

The SW-Synthesis module, using the *Timbre Model Editing* module, sees the current patch table through the *Patch Table Management* module that is part of the SW-Sequencer module. A synthesis module, using the functions given by SQAN, may create, delete, and modify the sounds in the patch table; in this way the sequencer simply acts as a data keeper. SQAN controls the timing process, reads the data from the MIDI sequences and knows when it is time to pass the right data to the DSP card. When the sequencer reads a timed MIDI data, it calls the SA.MIDI.DRIVER (Sound Accelerator MIDI Driver) function that parses the MIDI message and executes either the NOTE ON, NOTE OFF, or PROGRAM CHANGE modules. The first two modules realize the semantic translation from MIDI language to the DSP language, sending the appropriate data to the card. To find the right data, the third module has to ask the patch table management module to return the "timbre" data of the requested program-change entry while the sequencer is playing.

Timbre Control

The concept of patch is fundamental from the musician's point of view. It allows the musician to view every DSP synthesis module as a common MIDI synthesis device equipped with an editable li-

Fig. 2. Integration mechanism for adding new SPE modules.

SW-Sequencer

SW-Synthesis

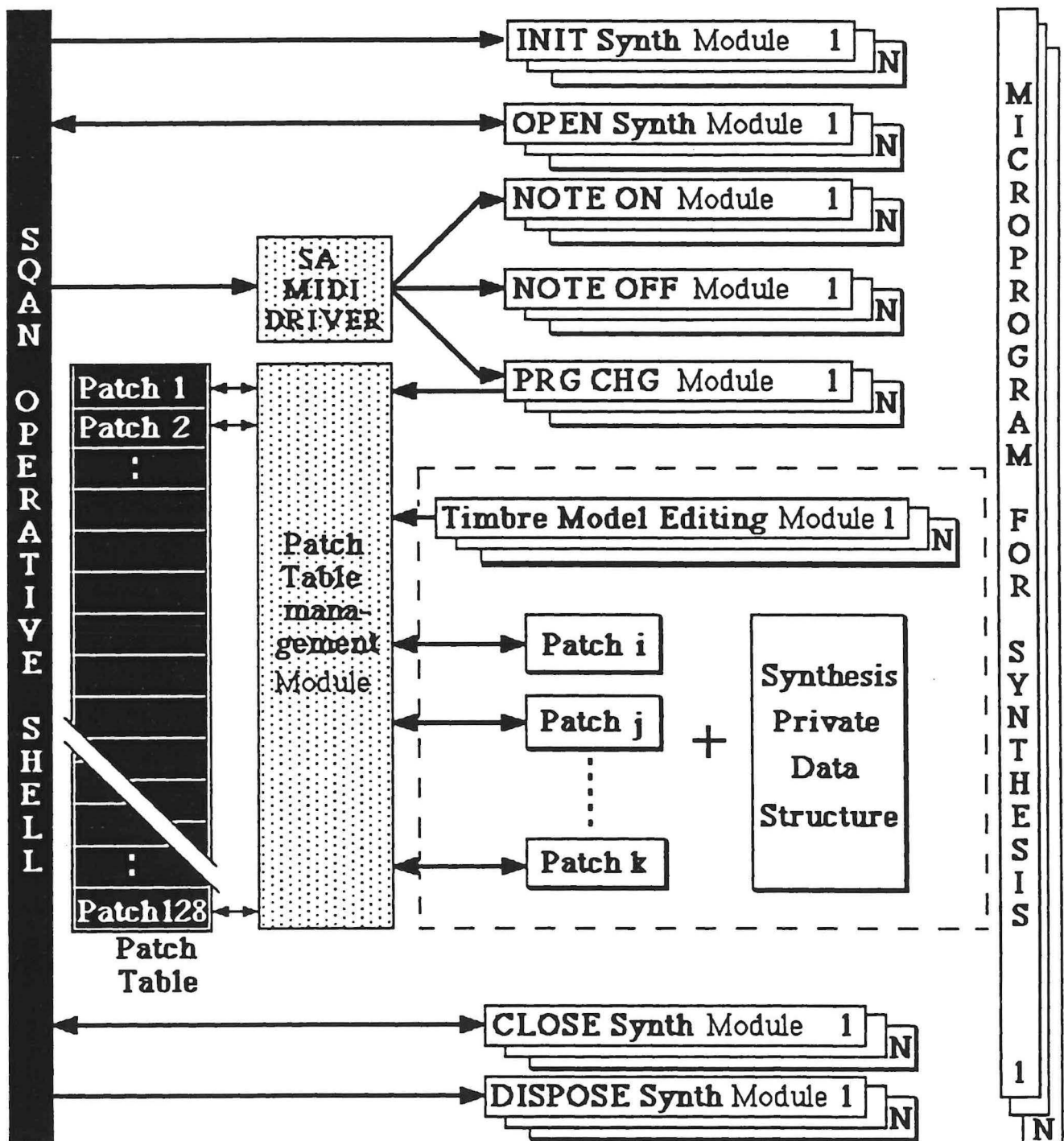


Fig. 3. The structure of SPE patch files.



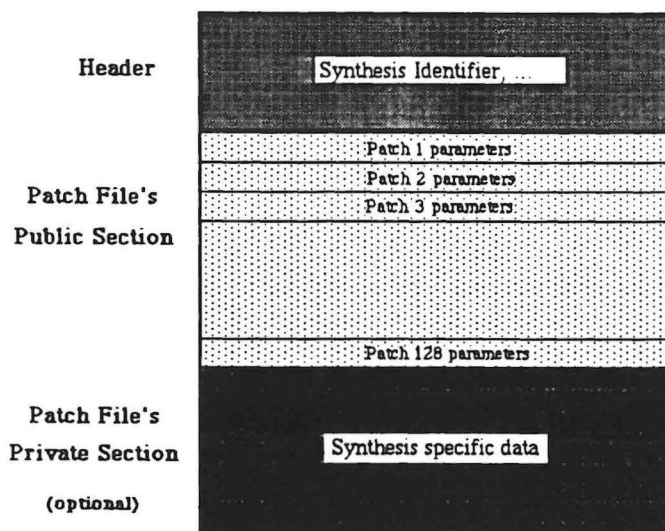
brary of sounds. Everyone should be able to open a song and a patch library from the "File" menu and let the sequencer run without special training. Even the musician who has no familiarity with the synthesis technique used, may insert or delete program-change commands and hear the result. A more advanced user will build his or her own libraries of patches in a consistent way for every synthesis technique available.

To maintain consistency, all patch files must have a common structure managed by the SOS. To use the whole potential of every technique, these files must also contain some data specific to each one. We ended up with the structure shown in Fig. 3.

A Patch File has three parts: *header*, *public*, and *private*. The header contains information used internally by the SOS; one relevant item is the identifier of the synthesis module to which the patch file belongs. The public part contains a sequence of parameter sets that describe a particular timbre; every set is a patch corresponding to a program-change number, and its elements may be edited by means of the interface functions of the related synthesis module; the public section of a patch file is available to both the SOS and the synthesis module. The private part of the patch file contains all the data that are meaningful for the current synthesis module (tables, samples, etc.); this part of the file is available only for the synthesis module and may be empty if it is not needed.

The problem is that the SOS does not know anything about the content of the part of the patch file which belongs to a particular synthesis technique (it does not even know if such a part is needed!), and that the synthesis modules have no direct means of accessing the whole file.

To solve the problem, a set of functions was created to manage the patch-oriented communication between the SOS and the synthesis modules. For instance, when a patch file is opened, the SOS first reads the header and the public part of the file, then it calls the *LoadStructure* function of the synthesis module which is defined by its identifier in the header. Every synthesis module programmer has to write such a function for reading the remaining part of the patch file in memory and interpreting its content as needed. When the sequencer finds a program-change command on a DSP track, it gets the location



of the corresponding set of parameters stored in memory and calls the *ProgramChange* function of the synthesis module, passing the value corresponding to the needed patch as a parameter. The *ProgramChange* function uses this value as a parameter to the *LoadPatchFromTable* function of the SOS to obtain a handle to the corresponding table of parameters. It is now able to retrieve these parameters and translate them in commands to be passed to the DSP card so the right timbre modification takes place. Similar communication protocols are used during the construction and the editing of the patch table.

These implementation choices add a bit of complexity to the implementation process of new synthesis modules, but it is quite transparent to the end-user, who will find a consistent environment for timbre management in every synthesis module implemented in SQAN. Moreover, the patch file structure allows different levels of access to the synthesis parameters according to the familiarity of the musician with the current synthesis technique.

Timbre Programming

Since the SOS does not make any assumption about the behavior of synthesis modules, any synthesis technique that allows real-time sound generation us-

ing the installed hardware may be implemented within the SQAN sequencer. We decided to implement synthesis techniques not easily available within commercial MIDI modules first. Currently there are three synthesis techniques integrated in SQAN: two-variable functions synthesis (TVF), waveshaping (nonlinear distortion), and LASy (Linear Automata Synthesis). Chronologically, TVF and waveshaping were the first sound modules added to SQAN. Programming the SQAN sequencer and the software modules for these synthesis techniques was carried out in the same period with tight collaboration among developers. A stand-alone version of the LASy algorithm has been independently developed at the L.I.M., and the integration of the LASy module took place later.

At least two software modules are necessary for integrating a new synthesis technique in the SQAN sequencer: a DSP56001 microprogram loaded on the Digidesign card, which computes the needed samples and sends them to the DAC on the card, and a driver, which translates the MIDI messages passed by the SOS into commands sent to the DSP card via its host interface. A third element, the interface module, is needed to attend to the communication with the user. This third module may vary greatly from one synthesis technique to another, but the user interface should be consistent with regard to patch management. From the musician's point of view, the SQAN sequencer and the new synthesis modules are a unique application, and even an occasional user should be able to control the routing of a sequencer track to the DSP by opening a patch library stored on disk.

It seems like a good idea to associate a stand-alone application dedicated to timbre research and patch building with every synthesis technique integrated in SQAN. This keeps the SQAN user interface more consistent and simpler while offering a more complete environment within the IMW for the musician who wants more control over the generated tones. Two such stand-alone applications are presently available: *Waver*, which covers TVF and waveshaping synthesis techniques along with wavelet transform, and the *LASy Workbench* for the LASy algorithm.

The following sections introduce the details of Wa-

ver, LASy Workbench, and the SQAN sound synthesis modules to give an overview of the whole IMW-SPE.

Waver

Waver is a workbench developed for experimenting with different kinds of time-domain sound synthesis. The method of synthesis by means of two-variable functions (TVF Synth is described in Borgonovo and Haus 1986) is a sound synthesis technique which is not commercially used, but with which we have been experimenting for many years at L.I.M. It consists of the sampling of a two-dimensional function over one orbit controlled by a small number of parameters. Waveshaping synthesis (WSH Synth described in Roads 1979) is a well known technique, which is also used in some commercial systems. It consists of the "shaping" of a sine wave, using it as argument of a given polynomial. The technique of synthesis by means of the wavelet transform (WLT Synth described in Kronland-Martinet 1988) was principally developed as an analysis technique similar to the Fourier transform. It can be used to synthesize sounds because, after transforming, many reconstruction rules exist that permit the resynthesis of the sound.

Waver is both a research and an execution application. It is a stand-alone application, but some of the TVF and WSH functions are common to the respective synthesis modules integrated in SQAN. Launching Waver, we can access the three main synthesis modules and some other useful features that can be necessary for realizing the whole process of sound production; this process goes from the creation of tables and waveforms, up to the listening and the real-time changing of the sound.

Implementation Notes

The TVF and WSH algorithms are completely realized using the DSP in order to minimize data exchange between the host computer and the DSP, thus reducing computing time. The host processor is involved only at the moment of sending new parameters.

Fig. 4. TVF Timbre Model
Editing dialog box.

$x(t) =$ $T +$ $+$ $* \sin($ $\pi * T +$ $\pi)$
 $y(t) =$ $T +$ $+$ $* \sin($ $\pi * T +$ $\pi)$
 $T = 1 /$ Freq.(Hz):
 $N.Samples:$ $Period(N.Samples)$

WLT synthesis has not been implemented in real time because a large number of samples have to go to and from the card. The sample swapping requires a lot of time and complicates the programming structure; this is due to the limited amount of memory on the DSP card, which does not allow a sound file to fit in the DSP card memory. This has shown us that the software architecture of SQAN is more suitable for the integration of algorithm-based synthesis techniques rather than synthesis techniques based on sample transformation that involve the transfer of a great amount of data between the host and the DSP card.

Software Modules

The TVF module permits the creation of TVF tables and the real-time experimentation of sound synthesis, using the dialog box shown in Fig. 4.

The WSH module allows the creation of tables of polynomials (Poly Tables) and the real-time audition of the synthesis results. The creation of a Poly Table may be realized by specifying the amplitude of the harmonics we want to be present in the sound result after the distortion. Then the program automatically computes the related polynomial coefficients according to the theory presented by Roads (1979). After this we can realize a sampling of this polynomial and group the polynomials in a table that may be stored and retrieved during the play phase. In this phase we can experiment with the waveshaping, choosing the polynomials and varying the parameters in real time via the dialog box shown in Fig. 5.

The WLT module provides for interactive realiza-

Fig. 5. WSH Timbre Model
Editing dialog box.

$PolyNum:$
 $y(t) =$ $Poly [$ $* \sin($ $\pi * T +$ $\pi)]$
 $T = 1 /$ Freq.(Hz):
 $Period(NSamples):$

tion (out of real time) of the wavelet-based synthesis cycle described by Kronland-Martinet (1988). We can start by loading a sampled sound, and then apply the wavelet transform, obtaining the modulus and phase of the transform for the voices as output. Using these voices we can start the resynthesis process of the sound while applying manipulations such as frequency transposition using Chebychev polynomials and cross-synthesis using data from different sounds. At the end of these operations we can store the result and listen to it by sending the stored sample file to the output DAC.

LASy

Linear Automata Synthesis (LASy) is a new sound synthesis technique (described in Chareyron 1990). The LASy algorithm sees a wave-table as a one-dimensional cellular automaton. The state of every cell of the automaton is computed periodically according to its transition rule. The corresponding values are sent to a DAC, allowing the generation of a vast variety of self-modifying tones.

The integration process of the LASy module into the IMW presents several interesting particularities. The programming of the SQAN sequencer and of the software modules for the other synthesis techniques was undertaken in the same period with tight collaboration between the programmers. On the contrary, the LASy Workbench was created as a stand-alone application and the integration began after the completion of SQAN. Furthermore, waveshaping and two-variable function synthesis are techniques that have many common characteristics,

while LASy is of a completely different nature. The discussion of the SQAN version of the LASy synthesis module will bring information of general value about the process of adding a new synthesis module to the SQAN sequencer environment. There are two applications present in the IMW which involve the LASy algorithm: the LASy module of the SQAN sequencer and the LASy Workbench, a stand-alone experimental program for this synthesis technique.

The LASy Workbench

The LASy Workbench is a software environment implementing the LASy algorithm. It is both a research and an execution application. The LASy Workbench allows musicians to experiment with sound synthesis and to build libraries of LASy software instruments to be used within the SQAN sequencer. It may be also used as a MIDI sound module with several real-time controls over the generated tones.

At launching time, default tables for waveforms and transition rules are automatically loaded so that the musician is immediately able to experiment with LASy synthesis. The sound generation is controlled by means of the computer keyboard (layered like a musical keyboard) or an external source (musical keyboard, sequencer, etc.) connected to the MIDI interface of the computer. The main screen shows most of the synthesis parameters which may be modified using the mouse or the computer keyboard. Files of waveforms, transition rules, and instruments may be moved to and from disk.

Implementation Notes

Several important decisions have to be made during the implementation. One must choose the bit size to be used during the computing part of the algorithm; ideally, one should use a word length equivalent to the resolution of the DAC, but for practical reasons we chose to use 12-bit resolution with linear interpolation. The synchronization between the computing of a sample and its transmission to the DAC must also be set. For musical uses the synthesis should

generate a large range of pitches, and it was necessary to desynchronize the computing of a new sample and the feeding of the DAC. The last choice is related to the computing resources used during the synthesis process; in the first phase, the 68000 processor of the Macintosh carried the duty of the computing part of the algorithm; then, to allow polyphonic synthesis, the weight of the computing process had to be transferred to the DSP56001.

Software Modules

The *synthesis module* takes care of the real time sound synthesis. There are several versions of this module, and the musician may switch instantly from one to the other (DSP microprograms are kept in the Macintosh memory as software resources). In the first version, the 68000 processor attends to the computing part of the algorithm. The only task of the DSP56001 is the transmission of the right sample (taking into account the current frequency), in the right format, to the DAC. This combination allows only monophonic real-time synthesis. Since the wave-tables are kept in the computer main memory, it is possible to get the real-time graphic display of the evolving waveforms. This facility is very helpful in the research phase, since it is easier to understand the behavior of an automaton when looking at it while it is working. In the other versions of the synthesis module, a monophonic one and a polyphonic one, the whole process of synthesis is taken over by the DSP. By now the polyphony is limited to four voices, but six- to- eight-voice polyphony should be possible with the same hardware configuration.

The *sound construction module* is used to build LASy software instruments; the musician operates upon the main display to choose a waveform, and one or more transition rules, and to set a group of parameters related to the synthesis. Some standard tables are built into the LASy Workbench, which also provides tools to construct other tables. Since there are a great many possible tables (especially the transition rules ones), and a great part of them leads to musically dull results, the program offers a group of algorithms able to construct interesting waveforms or transition rules tables. When using these

tools, the musician needs only to set the values of some parameters and let the program build the tables. Two other tools are available to edit the tables; a text editor lets the user change the value of every element of the table, and a graphic editor gives similar capability by means of classic graphical tools.

The *real-time control module* is provided so that the mouse may be used to modify in real-time some parameters of the tone during sound synthesis. Two kinds of action (corresponding to the horizontal and vertical movement of the mouse) are available at the same time, associated with a change in volume or in frequency, cross-fading between two transition rules, modifications of the current transition rule, etc.

Waveforms, transition rules, and software instruments are grouped into tables, a number of which are on line at the same time. Management of these tables, such as copying and pasting from one table to another, is taken over by the *librarian module*.

The SQAN Sound Synthesis Modules

In the following sections, we will examine how the three synthesis modules presented above have been integrated with the MIDI-DSP sequencer.

The TVF Module

A microprogram written in 56001 assembly language realizes the sound synthesis in real time by sampling a table containing a two-variable function along an orbit defined by 10 parameters that may be changed to vary the timbre. The orbit equations are:

$$\begin{aligned}x(t) &= f_x T + p_x + I_x \sin(\phi_x T + \psi_x \pi), \text{ and} \\y(t) &= f_y T + p_y + I_y \sin(\phi_y T + \psi_y \pi).\end{aligned}$$

Owing to the complexity of the algorithm, the synthesis technique has been implemented in a monophonic version only. In fact, we have found time constraints due to the fact that the algorithm must calculate a sample using less than 1/44,100 sec, that is, the time between two successive interrupts of the SSI port attached to the DAC of the Sound Accelerator card. To solve the envelope problem we have used a simple model with only three parameters, vol-

ume, attack rate, and decay rate; this has permitted the simplification of the microcoded version of the algorithm. It is relevant to say that the sample table is loaded on the card only at synthesis initialization time, and it is eventually saved in the private part of the patch file, while the parameters associated with program-change events (sent to the card while the sequencer is playing) are stored in the public part.

Microprogram

The SQAN version of the TVF microprogram first calculates the next orbit value using the current parameters communicated by the sequencer when the last program-change message has been encountered on a DSP track. After executing a two-dimensional table lookup in the TVF table, the microprogram updates the amplitude value according to the envelope form and the attack or decay rate of the current MIDI note event (the key velocity parameter). Finally, the new sample is sent to the SSI port that each 1/44,100 sec interval sends a value to the output DAC.

SA.MIDI.DRIVER

The driver translates the note-on, note-off, and program-change messages into host commands that trigger the suitable DSP actions. When a note-on is passed to the driver, it calculates the new value of the frequency parameter of the synthesis technique in order to produce a sound of the desired frequency; the new timbre parameters are then passed to the DSP via host commands. The amplitude value of the output signal is set from the key velocity of the note-on message and the envelope parameters are passed to the DSP too. If there is more than one DSP card in the system, this part of the driver has to control the dynamic allocation of DSP resources for sound production, keeping an image of voice allocation on the cards; if no voice is currently available, the DSP with the oldest sound is stopped to allocate the new sound request.

When a note-off is sent, the driver simply stops the current sound with the specified key number, if it is active. If there are two or more cards, the driver looks for the proper card to stop using its cards allocation image.

In the program-change case, the driver has to find the correspondent data (the Orbit parameters) via the Patch Table Management Module (handled by SQAN) and then pass them to the card. Note that in this part of the driver, only the changed parameters are sent to the card to make the orbits' transition as gradual as possible, while in the note-on part of the driver all relevant parameters are sent to the DSP at the same time.

Timbre Model Editing Dialog

Figure 4 shows the dialog box for TVF timbre model editing. Inside this dialog, beside the button labeled *ENV* that allows a user to access the envelope control dialog, are three other buttons, *Load Patch*, *Save Patch*, and *Edit Patch*, which allow the user to invoke the patch table management functions of SW-Sequencer to call and modify patches. Something like this must be present for any other timbre control interface on synthesis modules, so that the user always knows where to look for such buttons; it also represents an important communication point between SW-Sequencer and SW-Synthesis during the timbre controlling phase.

The WSH Module

In our implementation WSH is seen as a one-dimensional special case of TVF synthesis, where $x = i$ allows the choice of shaping polynomial p_i stored in row i of the polynomials table (see below) while

$$y(t) = I \sin(2\pi ft + p)$$

is the input wave to be distorted. We note that changing from one polynomial to another may generate a discontinuity, so we must do this operation only when a program-change has occurred.

The WSH synthesis microprogram is a restriction of the TVF one; the only difference stands in the table that has been created with different criteria. In particular, TVF synthesis, supplied by a "Poly Table," could generate waveshaping providing that we leave all the orbit parameters (see the orbit equations above and Borgonovo and Haus 1986) excepting the following three:

$$\begin{aligned} f_x &= i \text{ (polynomial integer index)} \\ I_y &\in [-1, 1] \text{ instead of } I_y \in \mathfrak{R} \\ \Psi_y &\in \mathfrak{R} \end{aligned}$$

Therefore, both TVF and WSH may be classified as nonlinear distortion (NLD) techniques; in the TVF case, predicting the output results is much more difficult, because input and transfer are both functions of two variables. Furthermore, in the TVF case the "wraparound" concept is relevant while it is not in the WSH case, since $I_y \in [-1, 1]$. With respect to the envelope control, we have used the same simple model as for TVF. As in the TVF case, the sample tables are sent to the DSP at open time and are stored in the private part of the patch file; these tables (WSH Poly Tables), together with the TVF ones, may be built using Waver. All the parameters associated with the program-change events are stored in the public part, one for each entry (0...127). Among these parameters, there is the *PolyNum* parameter that indicates which polynomial function of the Poly Table is to be used to produce the related sound; this is surely the most important parameter, because it may substantially change the harmonic content of the sound.

Microprogram

The SQAN version of the WSH microprogram first calculates the next "orbit" value using the current parameters, that is, the current value of the sine input function is added to the value of the polynomial displacement of the polynomial within the table. Then the microprogram executes a two-dimensional table lookup in the Poly Table and updates the amplitude level according to the envelope form of the current note. Finally, it sends the calculated sample to the SSI port which after an interval of 1/44,100 sec sends the sample to the output DAC.

SA.MIDI.DRIVER

The driver is very similar to the TVF one. When a note-on is sent to the driver, it calculates the new value of the frequency parameter (the "Time Divider" parameter), computes the new values of the timbre parameters, and sends them to the DSP card

via host command. The amplitude value of the output signal is updated from the key velocity of the note-on message, and the envelope parameters are passed to the DSP, too. When a note-off is sent, the driver simply stops the current sound of the specified key number, if it is active. In the program-change case, once the driver has found the corresponding data (the orbit parameters), it passes them to the card (note that only the changed parameters are sent to the card). If there is more than one DSP card in the system, they are handled as in the TVF case.

Timbre Model Editing Dialog

Figure 5 shows the dialog box interface for WSH timbre model editing. Within this, the button ENV allows the musician to access the envelope control dialog. To control and modify patches, the three standard buttons *Load Patch*, *Save Patch*, and *Edit Patch* are available.

The LASy Module

Microprogram

The microprogram for the LASy algorithm allows polyphonic sound synthesis (four voices in the present implementation). We were happily surprised to find that the polyphonic microprogram written for the LASy Workbench could be used within SQAN without any modification. The more complicated part of the work was the management of the patches' tables since every operation requires that several messages be sent to and from the SQAN kernel. This is the price we pay to give the musician a consistent interface for any synthesis technique, and it seems more than justified.

Every software oscillator is associated with two tables in the card memory (which represent the two most recent states of the cellular automaton). The main loop of the microprogram updates these tables according to the given transition rule (which may be different for every oscillator). This loop is interrupted at the sample frequency (44.1 kHz) to feed the output DAC via the SSI port. The interrupt routine gets the right sample from the tables according to the oscilla-

tor frequency (using linear interpolation to compensate for the finite length of these tables). The samples from every oscillator are scaled according to its current volume, then added and sent to the output. The main loop looks at flags indicating a new note (for loading a new wave-table and setting the current volume), a note-off message (switching to the decay transition rule) or a shut down (setting the volume of the oscillators to 0).

SA.MIDI.DRIVER

The driver translates the note-on and note-off messages into host commands that set the corresponding flags into the card memory. It also allocates the software oscillators, for which it uses a simple priority-based algorithm. If some oscillator is free, it is used to sound the next note. If there is no free oscillator, the algorithm "robs" the oscillator that entered its decay phase first. If no algorithm is in decay phase, the algorithm "robs" the oscillator that has sounded for the longest time.

The translation of program-change messages presents a more complex problem. A software instrument depends on a reduced set of parameters and two (or more) tables, one for the waveform and one or more for the transition rule. These elements should be transmitted to the DSP card at every program change, interrupting the computing operations for the duration of the transfer. This is not a problem for the parameters and the waveform table since their limited length does not cause relevant delay in the sequencer work. On the other hand, the transition rule tables are often sufficiently large to slow down sequencer operation if their transmission occurs too frequently. It may therefore seem wise to inhibit program-change messages if they cause a transition rule transmission. Practical experimentation has shown that there was no perceptible delay for current musical applications (for example, sending a program-change on every beat), so no limitation was included into the program, but the musician must be warned—if extremely frequent program changes are necessary, or if absolute time accuracy is needed, the patches used should not cause the repeated transmission of transition rule tables.

Fig. 6. LASy Timbre Model
Editing dialog box.

Timbre Model Editing Dialog

Communication with the user is via the main dialog box, from which the musician may modify the parameters of the synthesis algorithm and load (or save) patches and tables. This dialog box is shown in Fig. 6. Software instruments may be modified or created from scratch, but it is not possible to create or edit the waveforms and transition rule tables. Sound is constructed using the LASy Workbench. The tables needed for a musical work are then imported in SQAN, which supports "tuning" the patches.

The management of patch tables internal to SQAN and program-change operation follows the general rules described above. The LASy synthesis tools may be opened from the File menu by choosing a file of patches.

There are two kinds of file used in the LASy module of SQAN. Files of patches are internal to the sequencer and are the only ones needed to open the synthesis modules or to manage the program changes. The other files are common to SQAN and the LASy Workbench. They have to be built in the workbench and then loaded into the sequencer. There are three types of such files: waveforms, transition rules, and instruments. A file of instruments is nearly equivalent to a file of patches, and when the user loads this type of file into the sequencer, it is immediately converted internally to a patch table that may be used to insert program changes and saved on disk.

Conclusions

New music workstations built around the Macintosh or some other personal computer have been brought out recently by both commercial companies and research centers. Since we have no direct experience with most of them, we cannot present a comparative report here, but we will try to position the IMW-SPE among some of the most popular of them.

The IMW is a set of integrated musical applications, each of which answers a particular musical need. The role of SPE is to offer musicians the power and the programmability of DSP synthesis while keeping the "user-friendliness" of a typical MIDI environment.

Other music workstations have been designed as complete composition/synthesis systems built around a personal computer. Recent systems of this kind are the *IRCAM Musical Workstation* (Lindemann et al. 1991) and the *Interim DynaPiano* (Pope 1991). Both of these include software- and hardware-based sound synthesis capabilities and composition tools that make use of powerful object-oriented programming techniques. The IRCAM Music Workstation hardware adds a multiprocessor expansion card to a NeXT computer, while the software includes a custom operating system and a set of musical tools of various levels. The hardware part of the Interim DynaPiano is a Sun RISC workstation completed with DSP and DAC/ADC expansions, the software component consists of Smalltalk-80 object classes addressing the various areas of musical creation. These two workstations allow the implementation of virtually every synthesis technique controlled by MIDI at the cost of some programming work. However, they may not be immediately usable by the average musician, and their overall cost places them out of reach for many interested parties. There is also a complete music environment available for the Macintosh computer—the *Kyma* system from Symbolic Sound Corporation (Scaletti 1989). It is the combination of the Capybara DSP expansion hardware and the Kyma software environment, which acts as an object-oriented composition and synthesis tool.

More consumer-oriented products are also available for the Macintosh. Digidesign, Inc., has developed a popular line of hardware and software systems. The *Sound Tools* and *Pro Tools* packages are digital recording and processing systems not directly concerned with MIDI control. *SampleCell* (another Digidesign product) is a MIDI-controlled expansion card that can be used as a sample player. A similar product (limited to a ROM-based set of samples) is the *MacProteus* (by Digidesign as well). These two systems do not allow implementation of other synthesis techniques. There is a commercial sequencer, *Studio Vision* by Opcode, Inc., that uses the same hardware setup as the IMW-SPE (a Macintosh computer with a Digidesign DSP56001-based expansion card) to integrate MIDI sequencing with DSP sound generation. This product allows a MIDI sequence to drive the emission of previously sampled sounds by the DSP card, but it does not use the hardware as a note generator, as does the IMW-SPE.

Every product has its particularity and will fit the needs of some category of musicians. At L.I.M. we think there is still much to do in the field of sound synthesis, and our goal was to build a useful tool that combines the power of DSP and the simplicity of MIDI control. In addition, we chose not to develop custom hardware but use a widespread and reasonably priced hardware configuration (a matter of necessity), and we wanted to bring out an application that was well integrated with other work at L.I.M. With these guidelines in mind, we think the IMW-SPE is the better solution for us, as it could be for others in the computer music field.

The IMW-SPE is still a work in progress, so it has not yet been really used as a musical production tool. Preliminary testing has driven us to question some aspects of the original design. The most debatable point is the inclusion of a custom MIDI sequencer to drive the DSP modules. Most musicians like to work with their favorite sequencer, and very powerful ones are commercially available for the Macintosh. The integration level provided by the Macintosh's MIDI management tools allows the bypassing of the SQAN sequencer, so any application generating a stream of MIDI data is able to control the DSP modules. To this end we are working to drop the se-

quencer and rewrite the application as an independent MIDI-DSP driver. From the musician's point of view, every synthesis module of the MIDI-DSP driver will take the part of a MIDI expander seen as a MIDI device by the Macintosh MIDI manager. These modules will include the proper DSP microprogram and the interface routines which make them act as editor-librarians for the proper synthesis technique. As a further advantage of this solution, it will be easier to add other SQAN-DSP modules.

Many hardware constraints have limited the power of the IMW-SPE, but the actual trend in technology lets us hope for solutions in a few years. The most relevant constraints were the absence of DMA channels on the host computer, and the lack of sufficient memory for both microprograms and sound data on the DSP cards.

As a further improvement to our system we are working to give the musician the multitimbral feature of the DSP cards; currently, the musician can do polyphonic synthesis using from one to four DSP cards but only by means of the same synthesis module loaded on the different cards.

Availability

The software of the IMW-SPE is available to non-profit institutions by contacting the authors through Goffredo Haus, Scientific Director, Laboratorio di Informatica Musicale, Dipartimento di Scienze dell'Informazione, Università degli Studi, via Comelico, 39, I-20135 Milano, Italy; telephone 39-2-55006-338; electronic mail Music@imiucca.csi.unimi.it.

Acknowledgments

This project has been partially supported by the Italian National Research Council (grants 89.00031.69, 90.00678.PF69, and 91.00904.PF69) in the framework of the LRC C4 MUSIC: "Intelligent Music Workstation," Subproject 7 "Support Systems for Intellectual Activities," Finalized Project "Computer Systems and Parallel Computing."

Many thanks are due to Pier Luigi Bettitoni for development of the mixing feature and the MIDI file functions, and to the whole staff of the Laboratorio di Informatica Musicale for the scientific and technical support provided.

References

- Borgonovo, A., and G. Haus. 1986. "Sound Synthesis by Means of Two-Variable Functions: Experimental Criteria and Results." *Computer Music Journal* 10(3):57-71.
- Camurri, A., G. Haus, A. Stiglitz, and R. Zaccaria. 1989. "LRC C4 MUSIC: Architecture, High-Level Functional Specifications and Documentation Standards of the Intelligent Music Workstation." Technical Report 7/09, Computer Systems and Parallel Computing Project, Italian National Research Council.
- Chareyron, J. 1990. "Digital Synthesis of Self-Modifying Waveforms by Means of Linear Automata." *Computer Music Journal* 14(4):25-41.
- Kronland-Martinet, R. 1988. "The Wavelet Transform for Analysis, Synthesis and Processing of Speech and Music Sounds." *Computer Music Journal* 12(4):11-20.
- Lindemann, E., F. Dechelle, B. Smith, and M. Starkier. 1991. "The Architecture of the IRCAM Musical Workstation." *Computer Music Journal* 15(3):41-49.
- Loy, D. G. 1985. "Musicians Make a Standard: the MIDI Phenomenon." *Computer Music Journal* 9(4):8-26.
- Pope, S. T. 1991. "The Interim DyanPiano: An Integrated Computer Tool and Instrument for Composers." In *Proceedings of the Ninth Colloquio di Informatica Musicale*. Venice: Associazione di Informatica Musicale Italiana, pp. 70-79. (See also the article elsewhere in this issue of *Computer Music Journal*.)
- Roads, C. 1979. "A Tutorial on Non-linear Distortion or Waveshaping Synthesis." *Computer Music Journal* 3(2):29-34.
- Scaletti, C. 1989. "The Kyma/Platypus Computer Music Workstation." *Computer Music Journal* 13(2): 23-38. Reprinted in S. T. Pope, ed. 1991. *The Well-Tempered Object*. Cambridge, Massachusetts: MIT Press.
- Stiglitz, A., ed. 1991. "High Level Functional Specifications of the IMW Software Modules Developed at the State University of Milan." Technical Report 7/53, Computer Systems and Parallel Computing Project, Italian National Research Council.

