Rupert C. Nieberle, Stefan Koschorrek, Lutz Kosentzy, and Markus Freericks
CAMP
Technische Universität Berlin
Franklinstrasse 28
D-1000 Berlin 12
Germany
Nieberle@oppal.cs.tu-berlin.de

CAMP: Computer-Aided Music Processing

In the field of electronic music today, computers of many different types and sizes are in use. The musician is often hindered, however, by a steep learning curve, rigid system architecture, and limited application fields of the available music software. This document presents the design of a music development system that provides the committed musician with a creative tool for conventional music composition and production and for the realization of unconventional ideas such as algorithmic music or sound processing.

The CAMP-system (Nieberle et al. 1988) runs on personal computers linked by a network we have developed. A flexible, system-wide message-passing system for communication processes provides full network capability. Important system components, such as the output buffers and process table, are distributed. This architecture allows several musicians simultaneous utilization in a communicative, interactive way and increases the available computing power. More than one computer can control clusters of synthesizers simultaneously. The result is a distributed, real-time-oriented, interactive music development system that provides powerful tools easy to handle. The CAMP-system music software is written in the Forth programming language.

The CAMP-Forth music language includes several modern concepts and software engineering tools for easy development of music such as object-oriented elements (Nieberle and Orberger 1989) that provide a more abstract and therefore easier way to code compositional ideas and complex algorithms. One application of these tools is the implementation of device-independent drivers and on-line editors for MIDI-instruments and special DSP-hardware developed by our group.

The expansion of existing language elements for new musical composition tools is a straightforward process enabling us to build a multiparadigm system in the future. A built-in real-time LISP interpreter for interactive music processing has already been provided.

The "Forth Music System"

The underlying multitasking language for the "Forth music system" is based on the Forth dialect *ForthMacs,* written and distributed by Mitch Bradley. The CAMP-system uses parts of *Formula* the Forth Music Language (Anderson and Kuivila 1986a), a music system written in ForthMacs. Special operating system components for controlling MIDI devices and language tools for the formulation of musical sequences and events are provided.

Features of the Formula System

Formula provides a programming environment (Anderson and Kuivila 1986b; c; d) for the creation, manipulation, and control of musical events. All music generation and output is implemented via concurrent processes. Music is generated and played in real time under interactive control. Powerful means to articulate and synchronize music are built in. Because the system in Forth-based, users can expand the language. Finally, the user has full access to all levels of the Formula system, even the device drivers and other components written in assembly language.

Extensions to Formula

Introduction of Object-Oriented Language Features

Objects (Pountain 1987) make it easier to implement and handle complex musical processes. The

Computer Music Journal, Vol. 15, No. 2, Summer 1991,
© 1991 Massachusetts Institute of Technology.

mechanisms of data encapsulation and inheritance support stepwise system development and allow the development of concepts for the algorithmic generation of music in an evolutionary way. Standard Forth, however, does not support a data type concept.

Implementation of a Subset of the Lisp-Dialect "Scheme"

In many cases music can be represented or processed in a natural way with list-oriented languages. The ForthMacs process paradigm makes it impossible to treat a music score as a data object. Our Scheme-implementation Ylem (Freericks 1989) makes it possible to write programs that analyze inputs and generate output in a pattern-matching or rule-oriented way.

Tools for Handling and Integrating the Network

Access to the resources (output devices, file system, 'computing time,' etc.) of any other computer are available over the network (Koschorreck 1989; Kosensky 1989). It is possible to model complex communication structures using connections of software input/output ports.

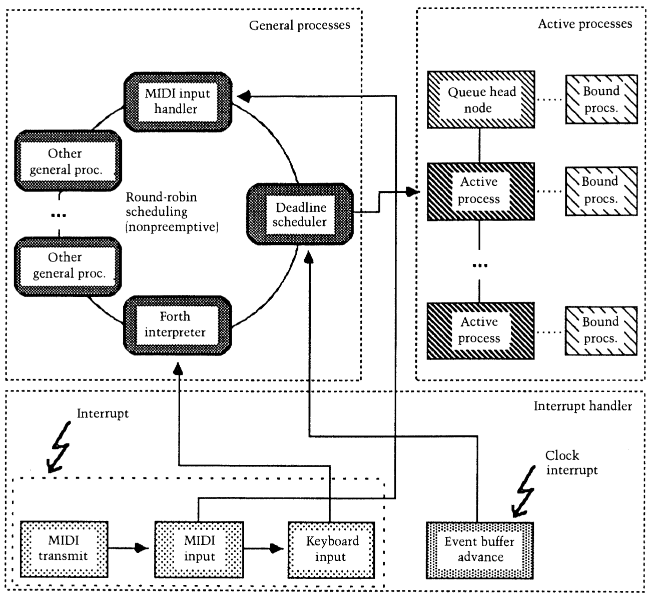


Fig. 1. The process structure of Formula.

Process Structure

Formula distinguishes between two basic types of process: the *general* and the *active* process. The general processes mainly accomplish tasks of the operating system, such as the managing of input devices (e.g., keyboard and MIDI input). The Forth interpreter is a general process, too. The scheduling of this process group proceeds by a round-robin strategy in a nonpreemptive way. We do not provide preemptive scheduling for processes as in the latest Formula version 3.4 (Anderson and Kuivila 1989) at present.

In the CAMP system general processes must call a special function at regular intervals to activate the next runnable process. This guarantees a fair distribution of computing time. General processes are usually deactivated. They are marked runnable by the corresponding interrupt handler only if they have something to do. The relations between the different processes and the interrupt handler are shown in Fig. 1.

The second type of process, the active processes, are the true event-generating elements of Formula. They are scheduled by a special general process, the "deadline scheduler." All active processes are held in a queue ordered by their deadlines. The "deadline" of a process is the moment when the process will generate its next event. The head of the queue is held by the process with the shortest deadline.

When control flows to the deadline scheduler, it activates this process and resumes the control as soon as the process has generated one event. Then the process is placed back in the queue according to its new deadline. After that, the procedure starts over again. The deadline scheduler resigns control to the next general process only if no active process is held in the queue or it is notified by a special flag that another process wants to run.

Every active process may have auxiliary processes (bound processes), which create additional parameters for event generation. They do not run by themselves, but are called in the course of activation of the active process they are bound to. In fact, they behave like coroutines rather than forming independent processes.

The Event Buffer

Generally, the creation of a single event in a musical piece cannot be done at the time of its output because its generation can demand considerable time. To separate the generation of events from their output (performance), Formula makes use of the so-called event buffer, a waiting queue. In the course of a musical piece, the deadline scheduler activates only that active process which will create the next temporal event. After it is computed, an event is inserted into the event buffer in a special form of representation—the event structure. The logical time of a process, that is, the time at which the performance of the event just computed will be inserted, will then be advanced and the next process is activated.

The global logical time is described by the *insert point,* which marks the place at which events are inserted into the event buffer. The insert point moves in jumps. The length of every jump depends on the distance to the next event to be processed.

Independent of the insert point, the *performance point* moves through the event buffer indicating real time. The event performance routine called by a timer interrupt at regular intervals takes each event at its current performance point and gives it to the appropriate device driver, which initiates its physical output. Subsequently, the performance point is moved to the next slot of the event buffer. The interval at which the event performance routine is invoked determines the minimal temporal resolution of the output.

Processes create events at some time before their scheduled performance. The temporal distance between performance point and insert point (called *buffer delay*) represents this interval. It must not be too long to ensure that interaction will come through within a short period of time. This delay time is adjustable.

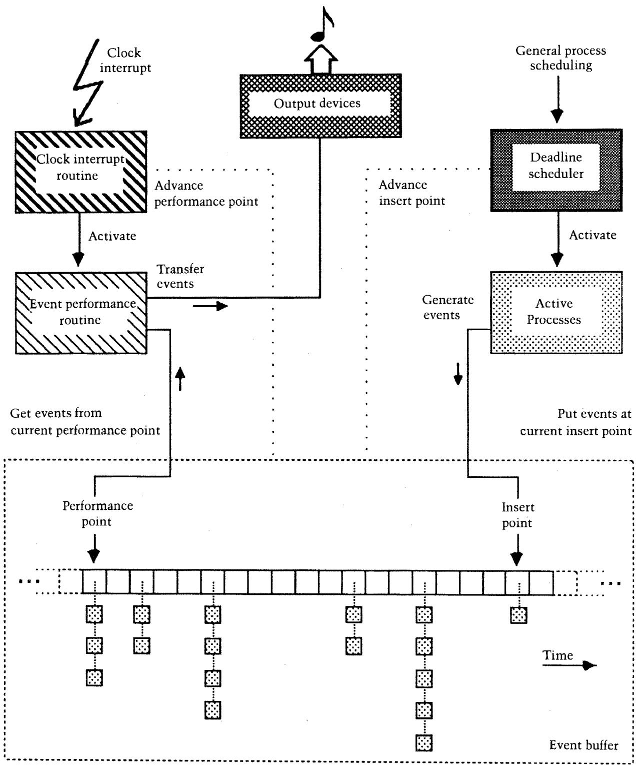


Fig. 2. Buffered output in the event buffer.

On the other hand, the average time for computing an event has to be shorter than the average distance between two events. The performance point would otherwise overrun the insertion point. If this happens, the event performance routine wil not be invoked long enough for the event processing to catch up.

Figure 2 shows the operations described above in simplified form. In particular, events pass through some intermediate states on their way through the event buffer. One of these states is the *future action queues.* With their help, events can be processed that have a performance time later than the current logical time of a process.

Network Integration

One of the central tasks in the course of distributing Formula in a network was the creation of an *event manager* that is placed between the active processes and the event buffer. As before, the active processes create a sequence of events, which now are not inserted into the event buffer directly, but are first directed to the event manager.

Using a *link-table,* this manager knows what device is connected to the currently active output channel of the process. A *device-table* specifies on what node in the network the appropriate device driver is installed. According to this information the event manager decides whether an event has to be directed to the local server or to a remote event server. In the latter case, the event manager on the destination computer receives the event from the network driver and hands it over to the event server there.

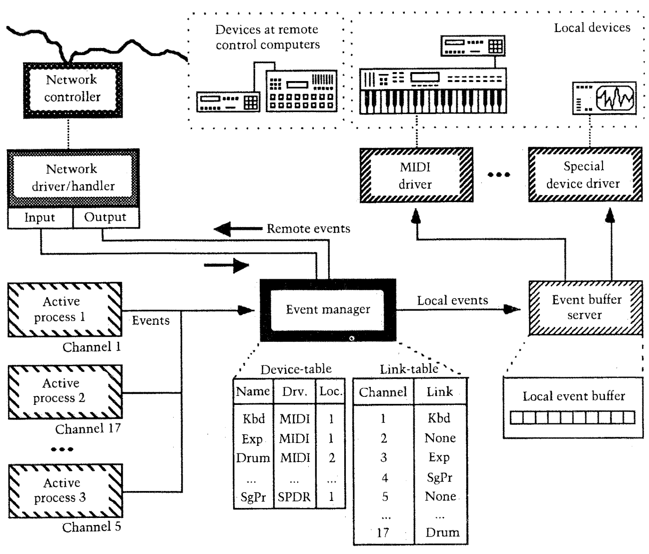
Such an event server manages a local event buffer working in the ordinary way but containing only events for output devices connected to the given local network node. Therefore, the local event buffers are completely disjunct and have to be synchronized. Although there is no global event buffer anymore, the local performance points are moving synchronously, building a global performance point. These relations are illustrated in Fig. 3.

The event manager simply redirects events, but lacks other process communication facilities. Further development of the event manager will lead to a "communication manager," which provides services such as bidirectional data exchange between processes. Again, this service should not depend on the physical location of the communication partners.

Output Devices

To connect various types of synthesizers, synthesis hardware manufacturers created the MIDI-standard in 1983. Basically, MIDI describes a serial interface protocol with a data transmission rate of 31,250 Baud. The protocol supports a channel concept. This allows the division of the data stream into 16 logical channels with different operation modes of the receivers. The commands available are divided into four classes: system exclusive, system common, real time, and channel commands. Data

Fig. 3. The network as a link between the generation and performance of events.



Device-table

Name	Drv.	Loc.
Kbd	MIDI	1
Exp	MIDI	1
Drum	MIDI	2
...	...	...
SgPr	SPDR	1

Link-table

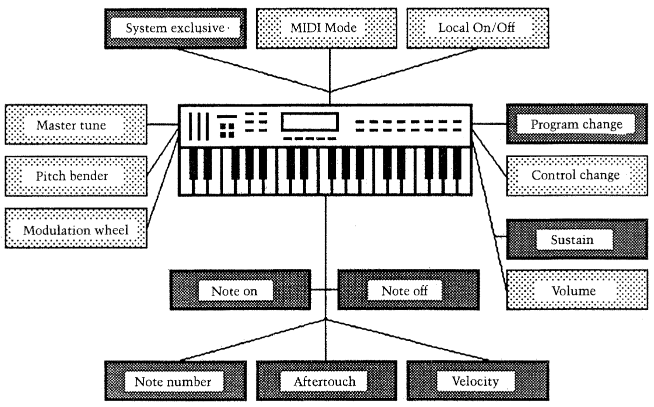
Channel	Link
1	Kbd
2	None
3	Exp
4	SgPr
5	None
...	...
17	Drum

are transmitted in bytes and structured by a distinction between command and data bytes.

Figure 4 shows the synthesizer parameters that are controllable by MIDI. The parameters that can be controlled by Formula are shaded darker in Figure 4. The main advantages of MIDI are, apart from its standardization, simple data handling and (as a result) low hardware costs. There are also some disadvantages. First, there are possible signal distortions as MIDI devices are connected in serial. The low transmission rate, which leads to audible delays when transmitting large data quantities, has to be mentioned as well. This Problem, called "MIDI overload," can be avoided with the CAMP-system by suitable distribution of MIDI devices among network nodes.

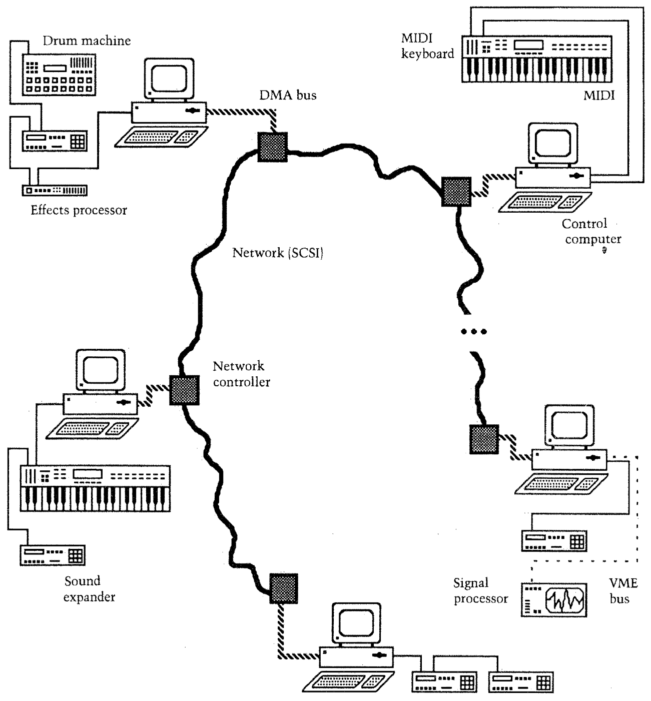
The original Formula version was consequently designed for use with MIDI devices. For example, the MIDI-typical representation of a single note by two separated events, "key-down" and "key-up," is found directly in the system architecture (in the future action queue for instance). Like Formula, the CAMP-system is based on the precondition that the performance of an event costs only a little computing time while its generation can consume a

Fig. 4. MIDI parameters (those in the darker boxes are controllable from Formula).



great amount. As a result of the modular system design, devices can be added and integrated in an easy way. Output drivers are not tied in firmly; rather, they are handled by the event manager as any other process.

Fig. 5. Exemplary configuration of a distributed music system.



A Multicomputer Music Development System

At this point, we will look more closely at the basic features of the distributed multicomputer music development system developed by CAMP. It is based on a homogeneous PC network containing Atari ST computers, which is accessible through a network operating system specially designed for musical purposes. The musical nucleus consists of a distributed version of Formula that has been modified to allow integration into the network. Furthermore, several composition-oriented language extensions based on objects (e.g., abstract data types, including heap- and list-oriented programming environment) have been integrated into the music processing system.

The system is written in Forth with minor portions of assembly language and therefore should be portable to other computers such as the Apple Macintosh or even a SUN workstation. An overview of the intended hardware architecture is given in Fig. 5.

Network Hardware

The control computers are connected with the network via their DMA-ports, which allow a data-transfer rate of up to 12 Mbit per second. The transmission protocol used is ASCI, a subset of the widely used SCSI protocol. SCSI is a universal interface developed in 1979 according to the IBM input/output concept, managing an 8-bit, bidirectional interface with a transfer rate of about 3 MByte per second maximum.

If the fast SCSI network-controllers developed by CAMP are not used, MIDI can be used as a substitute. In this case MIDI handles both synthesizer control and network communication via a special system exclusive protocol. Together with the slow data-transfer rate of MIDI, this will obviously result in decreased overall performance. For signal processing and sound synthesis a box with cascadable Motorola DSP 56001 boards was developed by CAMP, details of which are discussed in (Nieberle and Modler 1988).

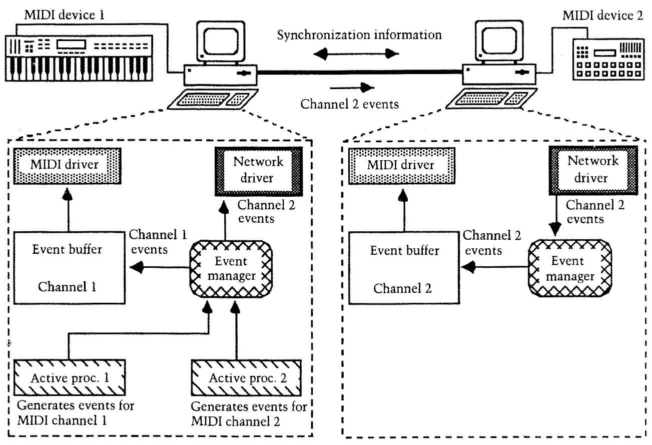


Fig. 6. Distribution of the event buffer.

Network Software

The system does not meet "hard real-time" requirements. Since delays below audibility level (≈2–3 msec) are musically tolerable, it is sufficient to ensure that the response time remains very short. Should it be impossible to fulfill this condition due to system overload, the synchronization will be preserved, keeping the musical output in time by suppressing some notes. This compromise allows the system to react in a musically-more-tolerable manner than losing control of timing.

Some data structures of the operating system (e.g., the process-table) and central components of Formula are distributed among the several computers in the network. Processes can be started, influenced, and stopped on remote computers as well as on the local machine.

Dynamic System Modification and Expansion

Large software systems, especially for the quickly changing music market, should be easily expandable to avoid any hindrance by unnecessary system limitations. Therefore, powerful mechanisms for the integration of new components must be provided. For example, the flexible, object-based driver concept of the CAMP-system allows easy installation of new devices. In this respect, the Forth implementation language proves to be of particular advantage; nearly all parts of the system may be changed quickly, some even at run time.

Global Device Management

All output devices are accessible from any computer in the network without considering which control computer they are actually connected with. To address devices, each of them has a globally known name. Using this technique, a high level of abstraction from physical locations, in combination with the efficient utilization of expensive resources, is achieved. Besides that, a solution to the MIDI overload problem is offered by distributing the existing synthesizers, expanders, and so forth among the control computers. This results in clusters of MIDI devices communicating via the fast network.

Although MIDI devices will probably retain their dominating position in sound generating, other kinds of devices such as signal processing units are supported.

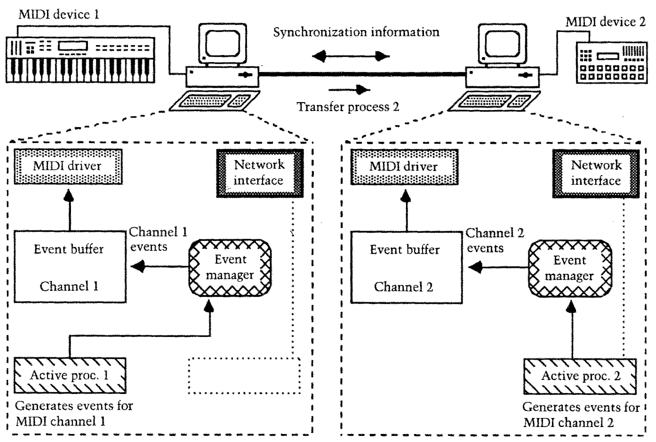
Load Balancing

To obtain maximum utilization of available capacities, two methods of load balancing will be provided. First, MIDI overload is prevented by switching to the network. As a second method, processes that require large amounts of compute time (e.g., graphics) and cannot be executed in time on one single computer are transferred to a remote computer with spare CPU time. Figs. 6 and 7 illustrate these mechanisms.

Inter-Process Communication

Process I/O takes place through *ports.* Each output port can be connected (linked) to the input port of any other process and vice versa. Complex communication models (similar to UNIX) can thus be managed. Default links help to minimize the required user action; e.g., note-generating processes

Fig. 7. Process migration in an overloaded situation.



are linked automatically to the appropriate MIDI driver. Output devices exclusively receive data in this model, whereas input devices are expected to only send data. As an additional advantage, many musicians are familiar with this technique from analog synthesizers or patch-bays.

Inter-process communication (Puckette 1986) and data transfer are performed through a message-passing mechanism that automatically transfers messages to remote processes. This way there is no need for the user to care about physical locations; the distributed character of the system is hidden. Nevertheless, explicit access to resources located on remote computers (e.g., the file system) is supported by special commands.

Automatic System Initialization and Configuration

On system startup, active computers and available I/O devices are automatically recognized. Any changes to this starting configuration at run time are taken into account. This feature is of special importance because at least some of the potential users are not experts and should not be concerned with problems of system administration. Besides this, music systems have to be as mobile as possible, so their setup should need a minimum of user activity.

Advantages of the Present Design

Our system gives the user the ability to use a great variety of resources that ensure maximum application potential and allow realization of new ideas beyond the standard. The programming interface, language, and underlying system are interactive. The real-time capabilities enable the musician to use the system like a conventional instrument; cooperation of several musicians is supported as well. Lastly, the patch model of interprocess communication is well known to many musicians.

Advantages to the Developer

The implementation language Forth permits extensive customization to user needs. The system architecture is flexible. Due to the hierarchical system structure, the desired degree of complexity can be selected by the user depending on his or her knowledge. The integration of several elements particularly suitable for musical work (e.g., object- and list-oriented elements, networking, distribution), derived from different languages and architectures, form a multiparadigm system.

System Availability

The authors have set up a non-profit organization for the purpose of distributing the CAMP software. They can be contacted for more information at either of the addresses below.

CAMP e.V.
Paul Modler
Liegnitzerstrasse 22
D-1000 Berlin 36
Germany

Rupert C. Nieberle
Skalitzerstrasse 62
D-1000 Berlin 36
Germany

References

Anderson, D. P., and R. Kuivila. 1986a. *FORMULA on the Atari ST.*

Anderson, D. P., and R. Kuivila. 1986b. "Accurately Timed Generation of Discrete Musical Events." *Computer Music Journal* 10(3): 49–56.

Anderson, D. P., and R. Kuivila. 1986c. "A Model of Real-Time Computation for Computer Music." In *Proceedings of the International Computer Music Conference.* San Francisco: Computer Music Association, pp. 35–41.

Anderson, D. P., and R. Kuivila. 1986d. "Timing Accuracy and Response Time in Interactive Systems." In *Proceedings of the International Computer Music Conference.* San Francisco: Computer Music Association, pp. 327–329.

Anderson, D. P., and R. Kuivila. 1989. *FORMULA version 3.4 Reference Manual.*

Freericks, M. 1989. "Erweiterung einer interaktiven Kompositionsumgebung um listenorientierte symbolische Sprachelemente." Studienarbeit, TU Berlin.

Koschorreck, S. 1989. "Entwurf und Implementierung einer verteilten Betriebssystemumgebung für interaktive Anwendungen mit Echtzeitcharakter auf Basis eines homogenen lokalen Netzes." Diplomarbeit, TU Berlin.

Kosensky, L. 1989. "Realisierung eines homogenen lokalen Netzwerks als Grundlage eines verteilten Mehrrechnersystems für interaktive Anwendungen mit Echtzeitcharakter." Diplomarbeit, TU Berlin.

Nieberle, R. C., et al. 1988. "The CAMP-System: An Approach for Integration of Real-Time, Distributed and Interactive Features in a Multiparadigm Environment." In *Proceedings of the International Computer Music Conference.* San Francisco: Computer Music Association.

Nieberle, R. C., and P. Modler. 1988. "An Open Multiprocessing Architecture for Realtime Music Performance." In *Proceedings of the International Computer Music Conference.* San Francisco: Computer Music Association.

Nieberle, R. C., and M. Orberger. 1989. "Abstrakte Datentypen in Forth." Lecture notes, TU Berlin.

Pountain, D. 1987. *Object-Oriented Forth: Implementation of Data Structures.* London: Academic Press.

Puckette, M. 1986. "Interprocess Communication and Timing in Real-Time Computer Music Performance." In *Proceedings of the International Computer Music Conference.* San Francisco: Computer Music Association.