

Computer Music Workstations I Have Known and Loved*

Stephen Travis Pope

The Nomad Group, *Computer Music Journal*, CNMAT

P. O. Box 9496, Berkeley, California, 94709 USA

stp@CNMAT.Berkeley.edu <http://www.cnmat.berkeley.edu/~stp/>

ABSTRACT: This paper introduces a set of design criteria and points of current debate in the development of computer music workstations. It surveys the systems of the last ten years and makes several subjective comments on the design and implementation of computer-based tools for music composition, production, and live performance. The intent is to focus the reader's attention on the issues of hardware architecture and software support in defining computer-based tools and instruments.

Introduction

At the 1984 ICMC in Paris, an integrated composer's workstation for software sound synthesis and production was demonstrated by PCS/Cadmus, GmbH (Cadmus 1985); the system consisted of an engineering-class desk-top UNIX multi-processing computer with very fast integer, floating-point, and I/O performance, generous RAM and disk memories, a state-of-the-art video monitor, and a "studio-quality" DAC/ADC subsystem, for the approximate cost of US\$ 35,000. The software included a Music V-family batch "sound compiler," and a suite of graphical and interactive tools based on the software practice of the day.

The current technology in workstation-based music systems—as described in several articles in *Computer Music Journal* 16:3 (Kahrs and Killian; Ballista et al.; Pope 1992)—offers configurations such as the "Interim DynaPiano," which is based on an off-the-shelf engineering-class multi-processing UNIX workstation with very fast integer, floating-point, and I/O performance, generous RAM and disk memories, a state-of-the-art video monitor, "studio-quality" DAC/ADC subsystems, and not-all-that-dissimilar software, all for the approximate list price of US\$ 35,000.

So what's changed? Workstation configurations for music composition and production have in fact changed less (qualitatively) than the radical increase in system capacity and performance over the last decade would lead one to expect (or hope). Why is this, and is it good news or bad?

This paper will address these questions, and present examples of components of modern workstation-based CM based on three recent articles in *Computer Music Journal*—on computer music (CM) workstations (Pope 1992), software languages for sound synthesis (Pope 1993b), and on sound file systems (Pope and Van Rossum). The issues of computer-based systems for composition and realization tasks in the studio, and for real-time live performance, will be discussed based on (Pope 1993c), and the current system-level definitions of "real-time synthesizer," "digital recording system," "compositional language," and "voice editor" will be presented (Pope 1993a). In the final sections, the paper moves to four currently raging debates: (1) using special-purpose DSP hardware vs. off-the-shelf host computers, (2) "consumer-class" PCs vs. "engineering-class" workstations as platforms, (3) special-purpose music ASICs vs. DSPs vs. general-purpose RISC architectures, and (4) the (developer and end-user) programming language issue (high- vs. low- level, general- vs. special-purpose, graphical vs. text-oriented UI, etc.).

Background

The computer music literature includes a rich legacy of many flavors of computer-based "music workstations." Several good surveys, taxonomies, and anthologies exist (see especially Gordon; Loy and Abbott; Pennycook; Pope 1993a, 1993c, 1994), and there is general (informal) agreement about the definitions of the various system categories. Nevertheless, there are still many open issues in the design of computer-based music tools, systems, and instruments, and the debate demonstrates the breadth of opinions about some of the basic concepts involved.

The criteria for comparing systems used in several of the taxonomies center around where digital sound samples are created and stored, and how their manipulation is controlled. Using a general-purpose computer of any kind for interacting with synthesizers imposes a model or paradigm as to how their synthesis algorithms or sample storage facilities are mapped and presented to the user. Synthesizers either directly generate sample streams, or process relatively static sample databases, under real-time unmoderated, interactive (or MIDI)

control. Software voice editors, sample dump and patch editors, and other software tools often use MIDI system exclusive messages to either download command scripts (algorithms) and parameter settings, or to upload and download sample data files from/to the (RAM and disk) memory of the host computer under some end-user GUI's control. The categories of "what to present the user with"—a programming language with musical structures and functions, an oscilloscope, a "sound laboratory" model, a tape-based production and mixing studio, a score editor/sequencer, a "configurable" real-time performance instrument, etc.—differ fundamentally in where sounds are stored, how they are generated and manipulated, and from what central point their processing is controlled (Pope 1993a).

There are many interesting configurations of computers, signal processors, large high-throughput memories, MIDI or other control I/O, and multi-channel DAC/ADC interfaces. Computer music workstations are based on off-the-shelf PC or workstation computers with some music-specific sound and/or control I/O facilities. These systems range from one-piece programmable sampler/synthesizers, to Mac+MIDI/Max+I/O, to UNIX multi-processor workstations with some mix of DAC/ADCs, DSP co-processors or MIDI interfaces. They frequently provide multiple levels of storage and of processing models, and offer different tool paradigms implemented in software on the same hardware.

The next section presents some of the history of the last decade's development in this area, and introduces several design criteria and architectural considerations. This is followed by a survey section and then the discussion of the topics of current debate.

CM Workstations

The development of interactive workstation-based CM systems started very shortly after personal computers with enough power and flexibility to support musical applications became available. By the mid-1970's, several teams had already demonstrated workstation-based integrated hardware and software tools for real-time software sound synthesis or synthesizer control (e.g., on Alto computers at Xerox PARC, or on early Lisp Machines at the MIT AI Lab). These systems were clearly computers (rather than being confused with the analog synthesizers of the day), but were also much different than the "popular" computer music systems of the era—large DEC or IBM mainframe computers.

During the 5 years from 1977 (introduction of the Apple II and several other popular microcomputers) until about 1982 (introduction of MIDI and several MC68000-based UNIX workstations), there was a wide variety of small (non-mainframe) computer music systems. These ranged from analog synthesizers controlled by small personal computers (e.g., Apple-II-based systems) to powerful bit-sliced digital synthesizers (Walraff). At that time, the lines between music production tools and performance instruments were not so clear, as was the differentiation between PCs and workstation-class computers.

The same 5-year time period saw the rise of the UNIX operating system, first on "mini-computers" (a concept that has disappeared altogether) such as the popular DEC PDP-11 series, and later on single-user workstations and larger machines (then called "super-mini-computers) such as the DEC VAX-11 series. Several groups were busily developing software sound synthesis systems based on UNIX during this time (Moore, Pope 1993b; Pope and Van Rossum). The 1984-era Cadmus 9000 system mentioned above used the UCSD CARL cmusic software-based sound synthesis and processing suite, and the csound file system (Moore). It had no real-time performance capabilities other than playing sound files from the hard disk, though MIDI was added to it later (Pope 1986).

Interim DynaPianos

The series of HW/SW configurations I call the *Interim DynaPiano* (IDP, Pope 1986, 1987, 1992) are also based primarily (though not exclusively) on UNIX workstations, and support varying combinations of off-line production and real-time interactive performance. The motivation for the development of the IDP family is the desire to build a powerful, flexible, and portable computer-based composer's tool and musical instrument that is affordable by a professional composer (i.e., around the price of a good piano or sophisticated MIDI studio). The hardware and low-level software of the system should consist entirely of off-the-shelf commercial components. The basic configuration of these systems is consistent with a whole series of tools based on the core technology described above. The motivations for this are outlined below.

The importance of using a commercial PC or workstation computer is simply wide availability. A powerful multi-tasking operating system with built-in lower-level signal- and event-handling drivers and support libraries is required so that the higher-level software components can be flexible, portable, and abstract. These high-level and front-end components must be written in an interpreted or rapid-turn-around incrementally-

compiled software development and delivery environment. If possible, the development language and environment should provide abstraction layers that mask platform OS-dependent driver details.

The use of a powerful and abstract central programming language integrated with the user interface “shell” is very important to the “dynamic” part of an IDP; the goal is to address the issues of learnability, scalability, abstraction, and flexibility, and provide a system that meets the requirements of *exploratory programming systems* or *interactive programming environments* as defined in (Barstow, Shrobe, and Sandewall) and elsewhere. The system should also be designed to support interchangeable (“pluggable”) interactive front-ends (e.g., graphical editors, or musical structure description languages) and back-ends (e.g., MIDI output, sampled sound processing commands, sound compiler note-lists, or DSP co-processor control). The components of such packages have been defined (e.g., Layer and Richardson), as: (1) a powerful, abstract programming language with automatic storage management, interpreted and/or incrementally-compiled execution, and a run-time available compiler; (2) software libraries including reusable low- and high-level modules; (3) a windowed interactive user interface “shell” text and/or menu system; (4) a set of development, debugging, and code management tools; (5) an interface to “foreign-language” (often C and assembler) function calls; and (6) a software framework for constructing interactive graphical applications.

The primary languages for early such systems were Lisp and Smalltalk, and to a lesser extent Prolog and Forth, because of several basic concepts. All four languages provide an extremely simple, single-paradigm programming model and consistent syntax that scales well to large expressions (a matter of debate). All can be interpreted or compiled with ease, and are often delivered with interactive development environments based on one or more “read-eval-print loop” shells.

The history of the various Lisp machines demonstrates the scalability of Lisp both up and down, so that everything from high-level application frameworks to device drivers can be developed in a single language system. Lisp is very much alive today, on both PC- and engineering-class hosts. The Smalltalk heritage shows the development of the programming language, the basic class libraries, the user interface framework, and the delivery platform across at least four full generations. Several current Smalltalk implementations—IBM, ParcPlace-Digital, Inc., Free Software Foundation—offer sophisticated development environments that is also portable to a diverse range of platforms, fast, and stable. These systems are largely source-code compatible with “standard” Smalltalk-80.

IDP-like Systems of the 1980s and Early 1990s

There has been varying progress in the evolution of each of the hardware and software components listed above between the systems developed in the early 1980s and those used today. Comments on several of the systems that have influenced the current design will follow.

The research systems that appeared along the coasts of the USA (in Cambridge and Palo Alto) in the mid-to-late 1970’s were generally programmed in non-mainstream languages and used non-commercial (i.e., not widely available) hardware. The advent of UNIX and the Motorola 68000 microprocessor lead to a plethora of commercial UNIX workstation computers in the first years of the 1980’s that went by the moniker “3M machines” —1 MIPS (million instructions per second) CPU performance, 1 Mbyte RAM memory and 1 Mpixel bit-mapped screen resolution. Several groups used these to develop *computer music workstations* or *intelligent composer’s assistants* at that time.

The *SSSP* synthesizer developed by Bill Buxton et al. at the University of Toronto in the late 1970s combined a Digital Equipment Corp. DEC PDP-11 computer with special user interface and sound synthesis hardware (Buxton et al. 1978). The synthesizer could produce up to 16 voices using various synthesis methods in real-time under the control of the PDP-11. The software was a broad range of UNIX-based C-language routines and applications that all manipulated the same event and voice data structures (Buxton et al. 1979). This configuration—a custom hardware digital synthesizer controlled by a general-purpose multi-user computer (Buxton et al. 1978; Walraff)—has disappeared today, or is a Macintosh with a DSP card such as the SampleCell a contemporary example?

The *CARL/cmusic* software distribution put together in 1982 by F. Richard Moore and D. Gareth Loy at the Computer Audio Research Laboratory (CARL) at the University of California in San Diego (UCSD) included comprehensive C language libraries for event and signal processing of all sorts, and the *cmusic* “sound compiler,” a simple, extensible member of the Music-V family (Moore). The integration and use of the tools is via the UNIX C-shell, C preprocessor, and C compiler. The simplicity, comprehensiveness, and extensibility

of the CARL software has made it the basis of several powerful computer music research and production tools on a variety of platforms, including DEC VAXen, Sun workstations, and NeXT machines.

Christopher Fry's *Flavors Band* (Fry) was a Symbolics Lisp Machine-based tool kit based on the notion of phrase processors that are manipulated and applied in a menu-oriented and Lisp-listener user interface. It used a pre-MIDI connection to a Fender Rhodes Chroma synthesizer for output and was used for a variety of styles and productions. Flavors Band phrase processors served as the models for several current interaction paradigms, such as Miller Puckette's *Max* and the MODE's event-generators and -modifiers (Pope 1992).

The *CHANT/FORMES* environment developed by Xavier Rodet et al. at IRCAM between 1980 and 1985 (Rodet and Cointe) used (ObjV)Lisp running (in various incarnations) on DEC VAX, Sun workstation and Apple Macintosh computers. The basic description and processing systems of the FORMES environment were extended by Macintosh-based graphical user interfaces, connection to the CHANT synthesis language, and the *Esquisse* composition system.

Kyma (Scaletti) is a graphical sound manipulation and composition language developed by Carla Scaletti. It is linked to the *Capybara*, a scalable real-time digital signal processor built by Kurt Hebel. The Smalltalk-80-based software tools that comprise *Kyma* present a uniquely abstract and concrete composition and realization platform. *Kyma* is one of the only full IDP systems (as defined above) delivered on a personal computer (i.e., PC or Macintosh); it is also a prime example of multi-language integration with Smalltalk-80; in it Smalltalk methods generate, download, and schedule microcode for the *Capybara* DSP, as well as reading and writing MIDI.

The recent *Common Lisp music/Common music* (clm/cm) system developed by William Schottstaedt and Heinrich Taube at the CCRMA Center for Computer Research in Music and Acoustics at Stanford University combines a NeXT "cube" workstation with an Ariel Corp. *Quint Processor* with five Motorola DSP56001 co-processors (Schottstaedt; Taube). The combined system supports signal synthesis and processing, score description and management, and MIDI capture and performance in a unified Lisp-based environment. This is a powerful state-of-the-art Lisp-based music system. More recently, it has been ported to other Common Lisp platforms, such as Intel-based PCs.

Eric Lindemann, Miller Puckette et al.'s *IRCAM Musical Workstation* (Lindemann et al.) used a Next "cube" with a DSP card designed at IRCAM and manufactured by Ariel Corp. that contained two Intel i860 floating-point signal processors and a Motorola DSP56001 for I/O. The higher-level software included integrated and stand-alone tools for performance and programming.

There are several non-examples deserve citation to further demonstrate the discussion of CM workstations. The various C/C++/Objective C-based systems such as that of the NeXT computer's Sound and Music Kits, the "standard" CARL environment, or the Composer's Desktop Project (Atkins et al.) do provide sound compilers, vocoders and sound processing tools, but fail to integrate them into a fully-interactive exploratory programming environment. The range of Macintosh-based MIDI packages includes flexible programmable systems for music (e.g., MIDI-Lisp or HMSL), but these address the event and event list levels only (often using MIDI note events as the only abstraction), and generally rely on MIDI's crude model of signals. Many other PC- and workstation-based signal-oriented systems generally fall into the categories of closed applications (e.g., the Studer/Editech *Dyaxis*, hard-disk recorders, *Music-N* compiler tool kits, or samplers), that are not programmable "composer's workbenches." Such applications must be provided in an open and customizable way for a truly general workstation.

Issues

The following sections will address several of the issues of current debate related to the topics introduced above. It should be stressed that these represent my personal opinions (with some backing from the references), rather than a well-founded general unified theory.

What are Computer Music Workstations?

Based on the criteria given above in the discussion of the design criteria for the Interim DynaPiano, I believe that extensibility and programmability are very important to any CM workstation. This definition does, however, exclude most of the products on the market that bear this moniker. I consider most of them to be synthesizers with built-in effects boxes and some degree of programmability. End-user extensibility can be directly related to the availability of a general-purpose programming language. "Closed" hardware/software systems that provide limited extensibility cannot be considered workstations according to this definition,

though they might be very good synthesizers, hard-disk recorders, or effects boxes. There are, of course, gray areas, in that some “dedicated” devices also provide a useful degree of programmability (e.g., the Eventide H3000), while some computer-based tools do not provide general-purpose programming languages (e.g., Macintosh + Max + MIDI).

General- vs. Special-purpose Platform

There are three architectural choices for digital music-processing hardware: general-purpose CPUs, special-purpose DSP CPUs, and music-specific ASICs. The relative power and cost of these three options is, however, a function of time, and all systems based on three have been in common use for at least ten years.

Over the first half of the 1990s, the power of general-purpose CPUs has increased dramatically; today’s RISC CPU is many times more capable than that of 1990—when RISCs were just coming into vogue. The relative performance of dedicated DSP CPUs and the production costs of ASICs have not—in my opinion—kept pace with this, so that ever more applications are solvable using general-purpose CM workstations rather than dedicated hardware (e.g., Freed, Rodet, and Depalle). There is, however, no guarantee that some change in ASIC costs (better/faster FPLAs, for example) will not change this situation at some time in the future.

Every computer is designed for some model of what the user’s program wants to do. Whether a general-purpose RISC or special-purpose multi-bus DSP is better for some given task depends on how similar that task is to those for which the RISC and DSP were designed. Some DSPs, for example, make excellent digital filters—due to their good multiply-accumulate performance with small parameter sets—but poor reverberators—due to their limited on-chip memory. I believe it is too complex an issue to state categorically that special-purpose DSPs (or general-purpose RISCs) will always be preferable for musical applications. I believe that, if we’re lucky, we’ll see great improvements in all three areas (ASICs, DSPs, and RISCs) over the next decade, and will be able to combine them into music-specific configurations more easily.

UNIX, MacOS and DOS as CM Platforms

There are all sorts of religious wars going on within engineering groups and on the market as to whether “professional” computer systems (e.g., UNIX workstations) are still significantly different (better) than “consumer” (e.g., Macintosh or DOS) personal computer systems. This distinction is getting more and more blurred with the porting of “high-end” operating systems (e.g., Sun’s Solaris or NeXTSTEP) onto Intel-based hardware. The “low end” hardware is also getting better, with the introduction of the Motorola/IBM/Apple PowerPC, the PCI bus, the 100 MHz (and correct!) Intel Pentium CPUs, and other recent developments.

It is my contention, though, that there is still an important difference between a “souped up” personal computer and an “engineering class” UNIX workstation. UNIX is a powerful multi-tasking, multi-user operating system that has a large collection of OS facilities as “standard equipment”—interprocess communication, TCP/IP networking, shared memory, sockets, memory mapping, multi-processor support, client/server window systems, etc. It remains to be seen how far Microsoft and Apple can go and retain their required backward compatibility. An “engineering class” workstation (or Macintosh) generally also includes as standard hardware many items that are third-party options on Intel-based computers: a network interface, “CD-quality” audio I/O, a mouse, a fast disk bus, etc. There is also an important current trend among most of the UNIX workstation vendors towards symmetrical multiprocessing; Sun, IBM, DEC, SGI, and H-P all offer SMP systems with 4 or more processors. It is well known that many popular CM applications can well profit from SMP (as in the 1984 Cadmus system, or the 1995 SPARC-based IDP).

It would be my hope that there will be some successor to UNIX (and Smalltalk) in the next decade, and that it will be based on a good foundation, rather than being constrained to be backwards-compatible to a PC OS. Layered micro-kernel architectures and message-passing OS kernels are being developed that will extend the true real-time facilities in new OS platforms.

Development Tools

Having good low-level and high-level development tools for music applications is another very important criterion. In this area there has been a significant merging between the PC-based and workstation-based systems. It is important that a CM workstation include a general-purpose (abstract, high-level, reusable, modern) programming language with general-purpose and music-specific application libraries. The last ten years has seen several music-specific languages come and go, and the popular ones are still either based on the Music V paradigm (Lansky; Moore; Vercoe), or on general-purpose languages with music-specific extensions or libraries (Honing; Jaffe and Boynton; Oppenheim; Pope 1992; Scaletti; Schottstaedt; Taube).

The bad news is many of us are still programming with depressingly low-level tools. C was designed for writing compilers and device drivers, not large programs. C++ is a mistake in progress. I believe we should all look for the highest-level language that is appropriate for the task (Freed). For some tasks this will be a low-level language such as C, for most others, high-level abstract languages such as Lisp, Prolog or Smalltalk are better choices. The same applies to software development environments (as outlined above), but I won't belabor the point any further here. As above, it is my hope that in ten years more of us have what some of us have today—good rapid-turn-around incremental development environments based on high-level languages.

End-user Screen- and Mouse-based GUI Paradigms

Disappointingly little has changed in the last decade in the area of graphical user interface (GUI) interaction paradigms. Ten years ago, people acknowledged the Xerox Smalltalk and STAR systems as the originators of all common mouse- and menu-based GUI techniques, and today Apple and Microsoft are in court over it, but the technology has improved little. The only important recent GUI technology is Apple's handwriting recognition software. This is also of interest to CM, but I have yet to see any concrete applications of it.

There are numerous voices more eloquent than mine calling for better GUI technology, but no general agreement on what we should replace our windows and menus with. Within CM, there are several interesting proposals for novel application GUIs, but the mainstream is still very much 2-D and based on relatively ancient models such as the tape recorder or mixing desk. I hope for web-based sound database browsers, hyper-media score editors, and automatic-learning AI-based programming tools in the future, and for a system with 3-D graphical and aural virtual reality "renderers" as the GUI paradigm.

Real-time Performance Interfaces

In this area as well, things are relatively stable (which can be taken as good or bad news). Most CM systems provide MIDI and stereophonic sampled sound I/O, and only a few support higher-bandwidth interfaces such as multi-channel (four or more) DAC/ADCs, ZIPI control, HTM, or other real-time interfaces.

The severe bandwidth limitations of MIDI are being alleviated by the interfaces that support several parallel MIDI channels from one computer, but there is still no general solution, other than an entirely new protocol such as ZIPI. The same applies to sound I/O; there are several systems that attempt to standardize multi-channel sound formats (Pope and Van Rossum), but little general agreement on one of them. This is, however, one of the most active areas of CM research and development, and I believe the can be sure that there will be major advancements on the coming decade in this area (see the comments above).

Conclusions

The more things change, the more they stay the same. This is not all bad, however. The last decade has indeed seen radical improvements in the price-performance ratio of affordable CM systems. To return to the example of the 1984 Cadmus 9000 (2 * MC68010/24 MB RAM/880 MB disk) and the 1995 SPARCstation-based IDP (2 * SPARC/96 MB RAM/4 GB disk), there is a performance ratio of about 200 (two hundred) between them for some basic tasks such as software sound synthesis or sound file mixing. Even though much of the software I use has stayed basically the same, there is a big difference between programming in Music-11 or cmusic in 1984 and programming in Smalltalk and Cmix today, just as there is between the UNIX (or MacOS) of yesteryear and today's model. My prediction is that ever more of us will be able to afford open, flexible, and powerful tools for music processing and performed in the years to come.

General-purpose computers with music-specific I/O and/or processing hardware are delivering some of the most powerful, flexible, and interactive performance instruments and production tools in use or development today. A high-powered PC or UNIX workstation with a DAC/ADC system, a DSP processor and MIDI or ZIPI interfaces, combined with a flexible, multi-level software suite, can represent a special class of tool for composers, performers, producers, and theorists. The next ten years promise to bring us whole new generations of digital music workstations.

In this paper, it is not my intent to complain, nor to say "I told you so," but rather to survey the progress of the last decade, and to discuss several areas that I would like to focus the reader's attention on. It is important to repeat that there are several good surveys and taxonomies that better introduce the topic of CM workstations, and that the references should be consulted to "flesh out" many of the remarks made here.

Acknowledgments

* Title with thanks to Andy Moorer.

References

- Atkins, M., et al. 1987. "The Composer's Desktop Project." *Proceedings of the 1987 ICMC*. San Francisco: International Computer Music Association. pp. 146-150.
- Ballista, A., E. Casali, J. Chareyron, and G. Haus. 1992. "The MIDI/DSP Sound Processing Environment of the Intelligent Music Workstation." *CMJ* 16(3):57-72.
- Barstow, D., H. Shrobe, and E. Sandewall. 1984. *Interactive Programming Environments*. New York: McGraw Hill.
- Buxton, W. et al., 1978. "An Introduction to the SSSP Digital Synthesizer." *CMJ* 2(4): 28-38. Reprinted in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press.
- Buxton, W. et al., 1979. "The Evolution of the SSSP Score Editing Tools." *CMJ* 3(4): 14-25. Reprinted in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press.
- Cadmus. 1985. "Cadmus Computer Music System Product Announcement." *CMJ* 9(1): 76-77.
- Computer Audio Research Laboratory. 1983. CARL Software Start-up Kit. La Jolla, California: CARL, University of California in San Diego.
- Fichera, S. 1991. "Machine Tongues XIII: Real-Time Audio Conversion under a Time-Sharing Operating System." *CMJ* 15(3): 27-40.
- Freed, A. 1992. "Tools for Rapid Prototyping of Music Sound Synthesis Algorithms and Control Strategies." *Proceedings of the 1992 ICMC*.
- Freed, A., X. Rodet, and P. Depalle. 1993. "Synthesis and Control of Hundreds of Sinusoidal Partial on a Desktop Computer without Custom Hardware." *Proceedings of the 1993 ICMC*. pp. 98-101.
- Fry, C. 1984. Flavors Band: A Language for Specifying Musical Style." *CMJ* 8(4): 20-35. reprinted in S. T. Pope, ed. 1991. *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*. Cambridge, Massachusetts: MIT Press.
- Gordon, J. W. 1985. "System Architectures for Computer Music." *ACM Computing Surveys* 17(2): 191-234.
- Jaffe, D., and L. Boynton. 1989. "An Overview of the Sound and Music Kits for the NeXT Computer." *CMJ* 13(2): 48-55. reprinted in S. T. Pope, ed. *The Well-Tempered Object*. Cambridge, Massachusetts: MIT Press.
- Kahrs, M., and T. Killian. 1992. "Gnot Music: A Flexible Workstation for Orchestral Synthesis." *CMJ* 16(3):48-56.
- Lindemann, E. et al. 1991. "The Architecture of the IRCAM Musical Workstation." *CMJ* 15(3): 41-49.
- Lansky, P. 1990. *CMix Release Notes and Manuals*. Department of Music, Princeton University.
- Layer, D. K., and C. Richardson. 1991. "Lisp Systems in the 1990s." *Communications of the ACM*. 34(9): 48-57.
- Loy, D. G., and C. Abbott. 1985. "Programming Languages for Computer Music Synthesis, Performance, and Composition." *ACM Computing Surveys* 17(2): 235-266.
- Moore, F. R. 1990. *Elements of Computer Music*. Englewood Cliffs, New Jersey: Prentice-Hall.
- Oppenheim, D. V. 1995. *DMIX Release Notes*. Internet ftp from ccrma-ftp.stanford.edu/pub/st80.
- Pennycook, B. W. 1985. "Computer Music Interfaces: A Survey." *ACM Computing Surveys* 17(2): 267-289.
- Pope, S. T. 1986. "The Development of an Intelligent Composer's Assistant: Interactive Graphics Tools and Knowledge Representation for Composers." in *Proceedings of the 1986 ICMC*.
- Pope, S. T. 1987. "A Smalltalk-80-based Music Toolkit." in *Proceedings of the 1987 ICMC*.
- Pope, S. T. 1992. "The Interim DynaPiano: An Integrated Tool and Instrument for Composers." *CMJ* 16(3):73-91.
- Pope, S. T. 1993a. "Music Composition and Scoring by Computer." (Invited chapter) in G. Haus, ed. *Computer Music and Digital Audio Series, Volume 9: Music Processing*. A-R Editions. pp. 25-72.
- Pope, S. T. 1993b. "Machine Tongues XV: Three Packages for Software Sound Synthesis." *CMJ* 17(2)
- Pope, S. T. 1993c. "Real-Time Performance via User Interfaces to Musical Structures." *INTERFACE* 22(3): 195-212.
- Pope, S. T. 1994. "A Taxonomy of Computer Music." Editor's Notes in *CMJ* 18(1):5-8 and 19(2):5-9.
- Pope, S. T., and G. Van Rossum. 1995. "Machine Tongues XVIII. A Child's Garden of Sound File Formats." *CMJ* 19(1):25-63.
- Rodet, X., and P. Cointe. 1984. "FORMES: Composition and Scheduling of Processes." *CMJ* 8(3): 32-50. reprinted in S. T. Pope, ed. 1991. *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*. Cambridge, Massachusetts: MIT Press.
- Scaletti, C. 1989. "The Kyma/Platypus Computer Music Workstation." *CMJ* 13(2): 23-38. reprinted in S. T. Pope, ed. 1991. *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*. Cambridge, Massachusetts: MIT Press.
- Schottstaedt, W. 1994. "Machine Tongues XVII: CLM--Music V meets Common Lisp." *CMJ* 18(2):30-37.
- Taube, H. 1991. "Common Music: A Music Composition Language in Common Lisp and CLOS." *CMJ* 15(2): 21-32.
- Vercoe, B. 1991. *Csound Manual*. Cambridge, Massachusetts: MIT Media Laboratory.
- Wallraff, D. 1979. "The DMX-1000 Signal Processing Computer." *CMJ* 3(4):44-49.