Real-Time Issues In Computer Music *

Roger B. Dannenberg, David H. Jameson

Carnegie-Mellon University
IBM T.J. Watson Research Center

Introduction

The fields of music and real-time systems have many common concerns, and there are many ways in which the fields might benefit from interdisciplinary research and an exchange of ideas. Time is a fundamental parameter in both music and real-time systems. Music is often concerned with ways of representing and structuring time, whereas real-time systems offer analytic techniques and engineering solutions for time critical systems. Computer music researchers have been guided by the problems of music to explore some interesting new approaches to representation, language, scheduling, and system organization. This paper surveys some of the interesting issues raised by music that can be addressed by real-time systems research. We also examine some of the ways in which sound and music technology can be applied by real-time systems researchers.

Characteristic scheduling problems

We begin by outlining some of the problems that are often found in music systems, and which tend to distinguish real-time music systems from other sorts of systems. First, music tends to be very bursty - musical events tend to cluster around downbeats, with relatively little work to be done in between. Thus, computer music systems typically have a large amount of idle time and frequent periods of momentary overload. Some systems try to precompute and timestamp data to minimize the amount of time-critical work to be done [1]. This has led to work on fast scheduling and dispatching [2]. Music systems usually have soft deadlines in the sense that data should be computed even if it is late, although late in a computer music system is often just a few milliseconds, after which time the effects of the lateness are perceptible. It is also the case that events cannot be scheduled too early. Having a note sound early is just as wrong as having a note stop late. Music is often predictable at least a short distance into the future and even though the computation load is bursty it is usually possible to anticipate how much and when computation will be needed. This opens up the possibility of advanced scheduling techniques that try to compute dynamically schedules that minimize timing errors. In general, music output is intended for human listeners; therefore, there is a perceptual basis for scheduling [13]. Timing or data errors that cannot be perceived are not important, and music systems often use human perception as a basis for performance requirements. Music is perhaps the most demanding medium in terms of timing. While video can often tolerate an occasional skipped frame, and lip-sync requires resolutions of tens of milliseconds, deviations of less than 10 ms can be detected in music performances. This is why it is important to synchronize video to audio, and not the other way around, a mistake often made.

Integrated real-time systems

It is not uncommon to combine expert systems, graphical monitoring, and music processing in a single real-time system [3]. For example, we have built computer systems that listen to a live performer and synchronize an accompaniment while automatically "turning pages" of music on a graphics display [2]. Here, the problems include scheduling tasks with a diverse range of scheduling requirements (typically on a single processor), and finding the right "glue" to interconnect various modules. Interface builders, object linking, and IPC mechanisms that support real-time systems are sorely needed. Music processing itself comes in different forms and places different kinds of demands on a system so audio output requires hard real-time scheduling to avoid glitches. Live audio processing (e.g. digital mixing) requires small communication packets with very low communication delays to avoid accumulating audio input-to-output latency. MIDI requires fast response to asynchronous input. Many systems must support all of

*Published in: *Proceedings of the Real-Time Systems Symposium,* Raleigh-Durham, NC, December 1-3, 1993. IEEE Computer Society Press, 1993.

these requirements in a single system [12]. Control updates in many real-time systems and in MIDI are not time-stamped, they simply take place as soon as the update arrives. This asynchronous approach can lead to non-determinism that is difficult to debug and can make output unpredictable. Audio filter updates, for example, sometimes must be sample-synchronous to guarantee filter stability. Some systems use time-stamping so that controls are delivered exactly at a predetermined audio sample. This is not always desirable, however, because a sample-synchronous control system inherits the same hard real-time constraints as the signals it controls. Systems that offer a range of options seem to be desirable. A whole range of issues regarding mixed event and signal processing remain to be explored.

Languages and system specifications

Expressive music control is very demanding; it leads to complex systems and requires advances in programming languages in addition to real-time operating system support. In other words, the programming problems of specifying intricate temporal behaviors sometimes dwarf the systems problems of scheduling or performance. Computer music researchers have probably devised more approaches to specifying temporal behavior than any other group [14], [15], [4]. Computer music systems sometimes use load-shedding techniques to (a) avoid overloading a bandwidth-limited MIDI channel, or (b) to best use limited audio processing resources. Load shedding usually means that MIDI notes must not be left on, and active signals must be carefully attenuated to avoid the pop that would occur if a process were simply terminated asynchronously. Thus load shedding implies additional work before the load is reduced. This is partly a scheduling problem, but we describe it here to emphasize that load-shedding requires domain-specific information from the system designer.

Music technology for real time systems

We now describe a few applications of music technology to real-time systems research. The MIDI protocol can provide a simple real-time serial communication link to external monitoring equipment, and MIDI recorders are available off-the-shelf. For example, we use one machine running an experimental system to generate 100 MIDI notes per second and we record and plot the events on a second machine. If the sending program is not scheduled appropriately, we can readily see (and hear) it. We typically run the generator on RT Mach [16], and then perform operations like dragging a window, forking a process, or running ftp, to see how much interference is generated by other processes and devices. An easy-to use and continuous monitoring system like this has turned up some interesting problems that were previously unnoticed. Several researchers have begun to investigate the use of audio, and in particular musical sounds, as a means of understanding programs [8], [7]. We have built a debugging environment called Sonnet [11] that consists of a visual programming language to define runtime actions that can be triggered by running code. Many of the actions produce sounds that allow programs to be monitored and debugged. One very difficult issue to resolve is how to provide appropriate sounds without being too intrusive.

Computer music is clearly a very useful testbed for real-time systems. Unlike the example above where sound is being used explicitly as a new tool to solve some other problems (monitoring and debugging in this case) we can also use music as a safe environment for experimenting with problems in real-time without the possible dangerous side-effects caused by a broken system. The real- time programming language ORE [5], [6], [9], although originally designed to control a juggling robot was found to be just as effective for building some music software [10].

Educational applications

Computer music using MIDI can be a good teaching tool. The effects of latency are very apparent on music data (for perceptual reasons), and music is a natural real-time application for motivating real-time topics. Students generally find music and sound an exciting domain, and a welcome change from the typical "glass tty" interface we see too often. Digital music synthesis hardware is relatively inexpensive.

Conclusion

Real-time systems are essential to computer music, and new developments in scheduling, operating systems, and languages are certain to find applications in music. At the same time, we believe the issues that arise in computer music should be of interest to designers of real-time systems. Computer music systems are demanding and complex, and require a large degree of flexibility to support the creative goals of musicians. Musicians and composers were working in the "real-time business" long

before the computer, and we think the art and science of music-making has much to offer.

References

[1] Anderson, D. P. and R. Kuivila, *Accurately Timed Generation of Discrete Musical Events*, Computer Music Journal, V10(3), pp 48-56, Fall 1986.

[2] Dannenberg, Roger B, *Current Directions in Computer Music Research*, pp 225-262, System Development Foundation Benchmark Series, Mathews, M. V. and J. R. Pierce (Eds), MIT Press, 1989

[3] Dannenberg, R. B., *Real Time Control For Interactive Computer Music and Animation*, The Arts and Technology II: A Symposium, N. Zahler (Ed), pp 85-94, New London, Conn., Connecticut College 1989

[4] Dannenberg, R. B., *The Canon Score Language*, Computer Music Journal, V(13)1, pp 47-56, Spring 1989

[5] Donner, M. D., Jameson, D.H., *A Real-Time Juggling Robot*, IEEE Proceedings Real-Time Systems Symposium, December 1986

[6] Donner, M. D., Jameson, D.H., *Language and Operating System Features for Real-time Programming*, Computing Systems, Vol 1(1), University of California Press, 1988[FrAlJa 91] Francioni, J.M., Albright, L, Jackson, J.A., *Debugging Parallel Programs Using Sound*, ACM Proceedings Parallel and Distributed Debugging, May 1991

[7] Francioni, J.M., Jackson, J.A., *Breaking the Silence: Auralization of Parallel Program Behavior*, Technical Report TR 92-5-1, Computer Science Department, University of Southwestern Louisiana, Lafayette, LA, May 1992

[8] Hotchkiss, R.S., Wampler, C, *The Auditorialization of Scientific Information*, ACP Proceedings SUPERCOMPUTING '91, November 1991.

[9] Jameson, D.H., *ORE: Programming Real-Time Applications*, 5th Workshop on Real-Time Software and Operating Systems, IEEE, May 1988

[10] Jameson, D.H., *Computer Music Performance as a Real-Time Testbed*, Real Time Programming, Proceedings of the IFAC/IFIP workshop, 1992. (Atlanta, Georgia, May 1991)

[11] Jameson, D. H. , *Sonnet: Audio Enhanced Monitoring and Debugging*, Proceedings, International Computer Music Conference, Santa Fe, NM 1991

[12] Puckette M., *FTS: A Real-Time Monitor for Multiprocessor Music Synthesis*, Computer Music Journal, V15(3), pp 58-67, Fall 1991

[13] Rubine, D and McAvinney P, *Programmable Finger-tracking Instrument Controllers*, Computer Music Journal, V14(1), pp 26-41, Spring 1990

[14] Schottstaedt, B, *Pla: A Composer's Idea of a Language*, Computer Music Journal, V7(1), pp 11-20, Spring 1983

[15] Scaletti, C., and Johnson, E, *An Interactive Graphic Environment for Object-Oriented Music Composition and Sound Synthesis*, Proceedings of the 1988 Conference on Object-Oriented Languages and Systems, pp 18-26, ACM 1988.

[16] Tokuda H., Nakajima T., and Rao P., *Real-Time Mach: Toward a Predictable Real-Time System*, Proceedings of the USENIX Mach Workshop, USENIX, October 1990