

Personal Effects: Weaning Interactive Systems from MIDI

Robert Rowe, New York University (robert.rowe@nyu.edu)

Abstract: Interactive music systems in early implementations usually made use of the Musical Instrument Digital Interface (MIDI) standard. The MIDI standard has been recognized since its inception to be slow and limited in its scope of representation [Moore 88]. Reliance on outboard MIDI gear has doomed a generation of interactive works to obsolescence as the requisite hardware becomes unavailable. Faster and cheaper machines have in recent years made it possible to perform analysis, synthesis, sampling, and effects on the CPU of a general purpose personal computer, simultaneously with the execution of control level software. This paper discusses the advantages, limitations, and technology of such an approach.

1 MIDI Love/Hate

MIDI was the dominant protocol for the design and implementation of interactive music systems for a long time, and its use still persists today. There were good reasons for this dominance, particularly at the end of the 20th century when these systems were first being developed. MIDI synthesis, sampling, and effects gear produced high-quality digital audio at an affordable price. Moreover, offloading the synthesis duties onto a dedicated piece of hardware freed the CPU of the computer to concentrate on control level analysis and composition.

On the other hand, MIDI transmission rates are too slow to handle large control streams. Synthesizer manufacturers retreated into bland repetitions of the same sampling-based architectures. The need to stimulate demand by introducing new models every few years meant that works written for commercial gear faced technical obsolescence in a very short period of time as specific boxes broke down and could no longer be replaced.

Small wonder, then, that composers have embraced a new generation of technology that allows the rendering part of interactive music systems (synthesis, sampling, and effects) to be handled by the CPU of the same computer calculating larger control structures. The most widespread instance of this phenomenon is the Max/MSP platform, which itself reflects the evolution of the technology: Max alone is a MIDI-processing environment, while the later MSP extensions incorporate real-time digital audio processing into the control-level structures organized by Max.

The strength of the concept has led to a wealth of alternatives, however. Miller Puckette's open source Pd system is itself a primary reference for the development of MSP [Puckette, Apel, & Zicarelli 98]. A phenomenal outpouring of open source digital audio applications is listed on Sourceforge [sourceforge.net], and distributed through the AGNULA project [www.agnula.org]. Beyond the fervor of commercial and individual developers, which has led to such influential packages as Cook and Scavone's Synthesis Toolkit (STK) [Cook & Scavone 99] and the CLAM library [Amatriain, Arumi, & Ramirez 02], there have been several efforts to standardize digital audio rendering protocols for various reasons.

2 Digital audio standards

The Open Sound Control standard (OSC) is one of the most direct approaches to resolving the networking and representational limitations of MIDI [Wessel & Wright 98]. Other platforms have been shaped by international standards organizations, or by their connection to existing specifications. Two of these are the Structured Audio Orchestra Language (SAOL) [Vercoe, Gardner & Scheirer 98] and JSyn [Burk 98].

Both are deserving of discussion for different reasons: JSyn because a working implementation exists and is in widespread use. SAOL is of particular interest because it is defined as part of the MPEG-4 standard, and it may come to pass that MPEG-4-compliant devices will implement some version of it as adoption increases.

JSyn, developed and distributed by SoftSynth and its lead programmer Phil Burk, is a unit-generator-based real-time synthesis library for stand-alone applications or Java applets. Units can be dynamically allocated and interconnected to support varying synthesis and processing topologies even as sound is being produced. JSyn applets are realized on a client machine through the use of a plug-in attached to the user's web browser.

While JSyn implements a unit-generator paradigm that is familiar from several previous synthesis languages, the SAOL specification is written as a high-level description of a language that has been adopted as part of the MPEG-4 standard [Vercoe, Gardner, & Schierer 98]. Soundball, Inc. of New York City has developed a compiler that implements SAOL as a plug-in to a web browser. The compiler takes SAOL instruments and compiles them into native instructions on the host processor of the client machine. Another implementation is sfront, a translator from SAOL to the C programming language written by John Lazzaro and John Wawrzynek. While sfront is more complete, generally available, and supported by an online tutorial book, it cannot run in real time as it must undergo the translation from SAOL to C. Once the resulting C program is compiled, that can produce real-time sound.

3 Navigating the choices

Given the embarrassment of choices, composers may be forgiven for hesitating over which way to turn. The author's solution has been to develop a personal library of audio sampling, synthesis, and effects routines in C++. While starting anew with such a project inevitably entails reinventing several wheels, it facilitates coordination with an existing control layer [Rowe 01] and seems the most expressive way to develop new ideas: a set of personal effects.

Moreover, some years of experience with this approach has led to an appreciation of MIDI despite its flaws. In particular, the signals coming from a MIDI instrument are often far faster and more accurate than that which can be produced by a real-time analysis of the corresponding audio signal. Pitch-tracking of a piano is unlikely to be better than the MIDI signal emanating from a disklavier anytime

soon. The combination of open source references into a personal library (itself open source) has proved a flexible and enduring way to incorporate the lessons from such hard-won experience.

References

- [Amatriain, Arumi, & Ramirez 02] Amatriain, X., Arumi, P., and Ramirez, M. (2002) "CLAM – yet another library for audio and music processing?", *Proceedings of the 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications*.
- [Burk 98] Burk, P. (1998). "Jsyn – a real-time synthesis API for Java", *Proceedings of the 1998 International Computer Music Conference*. San Francisco: International Computer Music Association.
- [Cook & Scavone 99] Cook, P., and Scavone, G. (1999) "The Synthesis ToolKit (STK)", *Proceedings of the 1999 International Computer Music Conference*. San Francisco: International Computer Music Association.
- [Moore 88] Moore, F. R. (1988). "The dysfunctions of MIDI", *Computer Music Journal*, 12(1):19–28..
- [Puckette, Apel, & Zicarelli 98] Puckette, M., Apel, T., and Zicarelli, D. (1998). "Real-time audio analysis tools for Pd and MSP", *Proceedings of the 1998 International Computer Music Conference*. San Francisco: International Computer Music Association.
- [Rowe 01] Rowe, R. (2001). *Machine Musicianship*. Cambridge, MA: The MIT Press.
- [Vercoe, Gardner, & Schierer 98] Vercoe, B., Gardner, W., and Schierer, E. (1998). "Structured audio: Creation, transmission and rendering of parametric sound representations", *Proceedings of the IEEE*, 86(5): 922-940.
- [Wessel & Wright 02] Wessel, D. and Wright, M. (2002) "Problems and prospects for intimate musical control of computers", *Computer Music Journal*, 26(3).