A New Network and Communications Protocol for Electronic Musical Devices

Keith McMillen, David Simon, David Wessel, and Matthew Wright

Center for New Music and Audio Technologies (CNMAT)
Department of Music
University of California at Berkeley
1750 Arch Street
Berkeley, CA 94709

Zeta Music Partners
G-WIZ
2560 9th Street, Suite 212
Berkeley, CA 94710

e-mail: mcmillen@CNMAT.Berkeley.edu, wessel@CNMAT.Berkeley.edu, matt@CNMAT. Berkeley.edu

Abstract

An inexpensive, moderate speed network protocol with bounded latency is proposed for communication among computers, electronic musical instruments, and related media devices. This protocol, that we call ZIPI, is presented in terms of the OSI layered network model. We describe a particular physical and data link layer for a token-ring network that supports a peer-to-peer architecture. We then describe, in some detail, the Music Parameter Description Language (MPDL), one of the application layers.

1 Introduction

While a number musical applications have been well served by MIDI, there remains considerable frustration with this primarily keyboard-oriented protocol. MIDI is not really a network. It has low bandwidth, which it uses to only 80% efficiency. MIDI has an awkward addressing scheme for controlling musical events and provides no mechanisms for high-level control. With these shortcomings in mind and with a desire to provide for musically expressive control that goes beyond the keyboard model and addresses alternate controllers, we propose the ZIPI network protocol for electronic musical devices.

The ZIPI network specification defines a collection of protocols that allow musical instruments, computers, synthesizers, mixing consoles, lighting controls, and other similar devices to communicate with one another. The ZIPI protocols conform to the "Open Systems Interconnection" (OSI) model developed by the International Standards Organization for separating computer networks into various conceptual layers [Tanenbaum 1989]. In this paper we will concentrate on one of ZIPI's protocol layers, the Music Parameter Description Language (MPDL) application layer, but a brief overview of the ZIPI physical and data link layers precedes. A more detailed presentation of ZIPI can be found in an issue of the *Computer Music Journal* [McMillen 1994, McMillen, Wessel, & Wright 1994, McMillen, Simon, & Wright 1994, Wright 1994a, 1994b, 1994c].

2 Physical Layer

The ZIPI physical layer is a token ring architecture that provides for a deterministic, low-cost, efficient, moderate speed (250K to 20M bits/second), and low latency network. Logically, ZIPI devices are connected in a ring, in which each device passes data to the next one around the ring. However, as shown in Figure 1, the user is spared the complexities of cabling the ring and each device is connected to a "hub" at the center of the star.

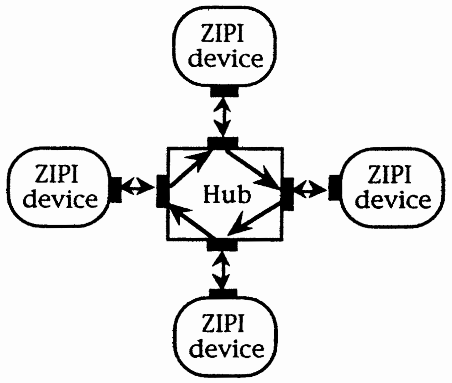

Figure 1
ZIPI's ring network architecture hidden in a hub

Devices are connected to the hub by a 7-wire cable with both directions of data flowing through it. The ZIPI physical implementation is synchronous and each connection carries a clock, data, and current line; the seventh wire shields the entire cable. ZIPI uses an opto-isolated current loop and either a 7-pin

DIN or 8-pin mini-DIN connector. The preferred implementation of a ZIPI network device is based on the 8530, an inexpensive serial communications controller chip available from Zilog and from AMD, or one of its close relatives. The 8530 running in "SDLC Loop Mode" implements most of the physical layer as well as some of the data link layer.

3 Data Link Layer

In addition to sending and receiving frames of data, ZIPI's data link layer establishes a unique address for each device, optionally sends acknowledgements of received packets, discards garbled packets by examining a CRC error detection checksum included with each packet, and negotiates with other devices on the network to determine the clock speed at which the network runs.

4 Application Layer

ZIPI contains several application layer protocols. These include the MPDL language for musical control, the MIDI protocol, and data dump protocols for samples and other forms of bulk data. We envision the definition of application layers for studio automation, lighting, and other media.

4.1 The Music Parameter Description Language (MPDL)

MPDL is for the control of music events. It encodes parameters and delivers them to notes or groups of notes on networked synthesizer(s). These parameters might come from a controller like a keyboard, a guitar controller system, or a computer. MPDL includes parameters that are well understood and universally implemented, such as loudness and pitch, with support for parameters such as brightness and other timbral representations that should be more common in the future [Wessel 1985]. A large number of parameters remain unspecified assuring expandability and flexibility.

The basic structure of an MPDL packet is shown in Figure 2. Keep in mind that some additional bytes are necessary for the lower layers of the network. For the ZIPI physical layer and data link layers 7 bytes are required, whereas an implementation of MPDL on Ethernet would require more.

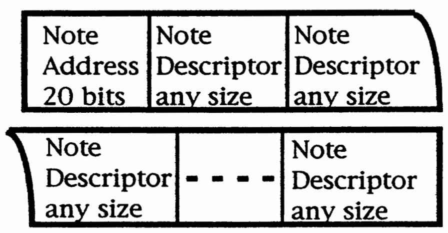

Figure 2
An MPDL packet

The note address indicates where the note descriptor data that follows will be applied. Packets can contain multiple note descriptors which helps cut down on network overhead. A list of some of the note descriptors for synthesis control is give in Table 1. The MPDL specification provides a similar list of controller note descriptors like key number, key velocity, percussion surface coordinates, and so on. We think it important to maintain a distinction between synth control parameters and measurements from controllers. A mapping function can be provided to link the two kinds of data.

A special note descriptor is reserved for a 32-bit time tag. This time tag is associated with the entire contents of a packet and can be used to assure greater timing accuracy and remove jitter in sequencers and MPDL data files that encode a performance or score. This is because controllers can time tag data nearer its source and synthesizers can use time tags to assure synchronization of events as in dense chords. For an explanation of how such time tags can be used in a responsive real-time system see Anderson and Kuivila [1986].

4.2 Address Space

A real difficulty with MIDI is that a note's address is its key number which indicates its pitch. Problems arise when a note's pitch changes over time or when there are two overlapping notes played on the same channel with the same pitch. In MPDL notes have individual addresses that are unrelated to their pitch or any other musical control parameter. A MPDL note number is just a number to identify that note, and an MPDL note number can have any pitch.

Another drawback of MIDI is that many of the controllers are associated with a MIDI channel rather than an individual note. For example, it is impossible to individually bend the pitch of one of the notes sounding on a given MIDI channel. You bend one and you bend them all.

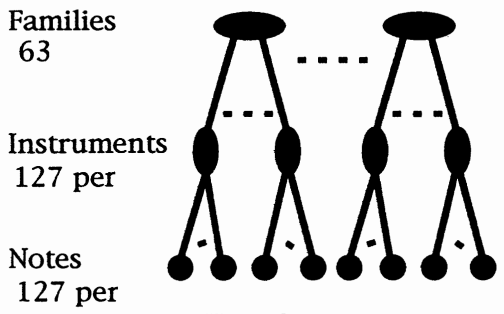

Figure 3
The MPDL hierarchy

The MPDL note address space is a three-level hierarchy. The names for the layers of the hierarchy fit an orchestral metaphor: There are "notes" within "instruments," and the instruments are grouped into "families." Any note descriptor message can be addressed to any level of the hierarchy: a stream of pitch messages could go to a single note that makes it bend independent of other notes on the instrument, while a loudness message could be used to control an entire family.

An instrument cannot change its family and belongs to only one family. The orchestral metaphor is just a metaphor; the purpose of a ZIPI address is to uniquely specify which note or group of notes a message applies to.

The device sending the note information must keep track of which notes are sounding and which are not, so that it can update parameters of already sounding notes. The device receiving the note information, of course, will not be capable of 1016127 (63 * 127 * 127) note polyphony, so it must allocate the available synthesis resources. It will also not be capable of storing parameters for 1016127 different notes. It is expected that ZIPI controllers will in practice only use a small subset of the address space; algorithms can take advantage of that expectation to store and record parameter values very quickly and without using very much memory.

4.3 Musical Control Parameters

After the note address has been chosen, a MPDL packet may contain any number of note descriptors intended for that address. A note descriptor gives a new value for a parameter, e.g., "pitch is B flat 2" or "pan hard left." The note descriptor consists of a note descriptor ID, which indicates which parameter is being updated, and some number of data bytes, which give the new value for that parameter. In Table 1 we show MPDL's current list of defined note descriptors. Each of the note descriptors has an 8 bit note descriptor ID that is omitted from the table.

Parameter	Combining Rule	Bytes
Articulation	``and"	1
Pitch	add	2
Frequency in Hertz	overwrite	4
Loudness	multiply	2
Amplitude	multiply	2
Brightness	multiply	1
Even/Odd Harmonic Balance	multiply	1
Pitched/Unpitched Balance	multiply	1
Roughness	multiply	1
Attack character	multiply	1
Inharmonicity	multiply	1
Pan Left/Right	multiply	1
Pan Up/Down	multiply	1
Pan Front/Back	multiply	1
Spatialization distance	multiply	2
Spatialization azimuth angle	add	1
Spatialization elevation angle	add	1
Multiple output levels	multiply	2
Program Change Immediately	overwrite	2
Program Change Future Notes	overwrite	2
Timbre space X dimension	add	1
Timbre space Y dimension	add	1
Timbre space Z dimension	add	1

Table 1
MPDL's defined note descriptors

4.4 Articulation

If the note descriptor ID is "articulation," the two high-order bits of the data byte specify one of the three articulation types given in Table 2

High-order bits	Articulation type
11	Trigger
10	(not used)
01	Reconfirm
00	Release

Table 2
MPDL Articulation Types

Trigger messages start a note; the new note will have any parameters that were set for that note before the trigger message was sent. Pitch and loudness are not part of the trigger message, so they should be set to the desired levels either before the trigger message is sent, or in the same MPDL packet as the trigger message. The order of the pitch, loudness, and trigger note descriptors within a ZIPI packet does not matter, all the updates in the packet are made prior to the trigger.

The note then sounds until a release message is received. If a new trigger message comes before the release message comes, the note re-attacks with no release. This is useful for legato phrasing.

A note retains its parameters after a release message; receipt of a new trigger message will articulate a new note with the same parameters as before.

Low-order bits	Behavior
000001	Release the note naturally
000010	Instantly silence the note
000011	Release the note naturally, unless it's still in the attack portion of the tone, in which case complete the attack portion and then release naturally

Table 3
Types of Release Messages

The remaining 6 bits of the articulation data byte specify exactly what kind of articulation occurs. In music, "articulation" can mean a lot more than "on and off." There are a large number of instrument-specific articulation styles, e.g., hammer-ons for guitar, lip slurs for brass instruments, and heavily tongued attacks for reed instruments. The problem with encoding these articulation types is that they are meaningful only in the context of certain instruments; it's difficult to say how to implement a hammer-on on a clarinet. We have specified some release types in Table 3. We are working to define more abstract articulation categories, expressed in a way that doesn't refer to a particular instrument. Hopefully these will be in a future version of MPDL as the possible values for the remaining six bits.

4.5 How the Levels Interact

What happens if the user sends an amplitude of 1 to a note, then an amplitude of 10 to the instrument containing that note, then an amplitude of 100 to the family containing that note? What's the actual amplitude of the sound produced?

There are four different ways to combine parameter values passed to different levels of the hierarchy. Each parameter uses one of these four rules. They are: "and," "multiply," "add," and "overwrite." A column in the Table 1 of note descriptors tells which of these four rules each parameter uses.

Only articulation uses the "and" rule, which is described in the detailed specification [McMillen, Wessel, & Wright 1994].

Most parameters use the "multiply" rule, meaning that each level of the hierarchy (notes, instruments, and families) stores its most recent value for the parameter, and the actual value that comes out is the product of these three numbers.

Amplitude is an example of a parameter with this rule. If two notes of an instrument have amplitudes of 20 and 10, they will have a relative amplitude ratio of 2:1 no matter how high or low the instrument's amplitude gets.

Note that what are being multiplied are offsets from a base value, and that the "base value" depends on the particular patch being played on the synthesizer. A flugelhorn will naturally be less bright than an oboe, so the mid-scale brightness value for a flugelhorn will produce a much less bright sound than the mid-scale brightness value for an oboe.

The "add" rule is just like the "multiply" rule, except that the three values for the parameter are added together instead of multiplied together. Pitch, for instance, is taken as an offset from middle C, and the offsets accumulate additively. If a family receives a pitch message of hex 7f00, which would be middle C#, the effect will be to transpose everything played by that family up a half step.

Acknowledgements

This work was supported in part by Grant C92-048 from the California State Department of Commerce Competitive Technologies Program. We would also like to thank the following individuals for their valued comments, sometimes very critical, regarding the ZIPI and MPDL specification: Jim Aiken, David Anderson, Marie-Dominique Baudot, Don Buchla, Richard Bugg, Tim Canning, Chuck Carlson, Lynx Crowe, Rob Currie, Steve Curtin, Peter Desain, Kim Flint, Adrian Freed, Guy Garnett, Mark Goldstein, Henkjan Honing, Dean Jacobs, Michael Land, Michael Lee, Carl Malone, Bill Mauchly, Peter McConnell, F. Richard Moore, Chris Muir, David Oppenheim, Stephen Pope, Rob Poor, Miller Puckette, John Senior, Warren Sirota, John Snell, Michael Stewart, Tovar, David Zicarelli.

References

[Anderson & Kuivila 1986] David Anderson & Ron Kuivila. Accurately Timed Generation of Discrete Musical Events. *Computer Music Journal,* 10(3): 48-56. (1986)

[McMillen 1994] Keith McMillen. ZIPI—Origins and Motivations. *Computer Music Journal,* 18(4). (in press)

[McMillen, Simon, & Wright 1994] Keith McMillen, David Simon, & Matt Wright. ZIPI Network Summary. *Computer Music Journal,* 18(4). (in press)

[McMillen, Wessel, & Wright 1994] Keith McMillen, David Wessel, & Matthew Wright. ZIPI's Music Parameter Description Language. *Computer Music Journal,* 18(4). (in press)

[Wright 1994a] Matthew Wright. ZIPI Examples. *Computer Music Journal,* 18(4). (in press)

[Wright 1994b] Matthew Wright. MIDI/ZIPI Comparison. *Computer Music Journal* , 18(4). (in press)

[Wright 1994c] Matthew Wright. ZIPI Frequently Asked Questions. *Computer Music Journal,* 18(4). (in press)

[Moore 1988] F. Richard Moore. The Dysfunctions of MIDI. *Computer Music Journal,* 12(1). (1988)

[Tanenbaum 1989] Andrew Tanenbaum. *Computer Networks* , Englewood Cliffs, New Jersey: Prentice Hall (1989)

[Wessel 1985] David Wessel. Timbre space as a musical control structure In Roads, C. and J. Strawn, eds. *Foundations of Computer Music*, Cambridge: The MIT Press. 640-657. (1985)