MIDI SHARE

A real time multi-tasks software module for Midi applications

Yann Orlarey et Hervé Lequay
GRAME
6 quai Jean Moulin BP 1185
69202 LYON CEDEX 01
Tel (33) 72.07.37.00 Fax (33) 72.07.37.01
e-mail : GRAME@rd.grame.fr

Abstract *: This paper introduces MidiShare, a real time software module meant for the development of Midi applications in a multi-tasks context. MidiShare brings facility in most of the fields concerned with the development of musical applications: communications management, precise time control, tasks scheduling. The originality of MidiShare lies in its capacity to deal with multiple Midi applications running at the same time. Besides, MidiShare offers some interesting possibilities, particularly the one consisting in a dynamic connection between Midi applications through internal links. MidiShare was at first designed for the Macintosh under MultiFinder, it is now available on other machines.*

Introduction

The availability of multi-tasking operating systems such as Multi-Finder, Unix and OS/2 on microcomputers is a decisive advantage in many application areas and offers real gains in term of power and ease of use.

Unfortunately, until now musicians have hardly benefited at all from the advantages of such systems due to the lack of music applications capable of running simultaneously in real time on a single machine. The internal layers of a Midi program often make use of critical resources of the computer such as timers, interrupts and serial link controllers which cannot be directly shared by more than one application, in addition, even multi-tasking operating systems are not generally a guarantee that applications run in real time.

Midishare came into existence from this observation with the ambition to provide the frame needed to develope musical applications in a multi-tasks and real time context. When defining MidiShare, we have tried to conciliate three aims :

- To propose solutions to the problems regularly met with in the development of all Midi applications.
- To permit the simultaneous running of these applications, especially as regards real-time routines and Midi communications.
- To allow for a cooperation between separately developed applications.

We are going to see how the various tools and services proposed by MidiShare comply with these objectives.

Events Managing

The whole system is based on the notion of event. An event is a dated entity possessing the informations for its own execution. The applications developed with MidiShare heavily relie on events to transmit and receive Midi messages but also to remember the tasks to be carried out.

It is generally not possible to use the host Operating System Memory Manager (MM) to deal with the allocation-deallocation of these events, since it revealed unadapted to a real-time context, non reentrant and inefficient, in terms of memory space when dealing with small zones of a few bytes.

To make up for these lacks, MidiShare has its own dynamic memory allocation module, adapted to real-time constraints. For example, some functions permit to create or delete events (under interrupt too) with an extremely brief and constant time response. Moreover, these events can be accessed to and handled in a uniform and orthogonal way by a small number of routines which take automatically in charge the various memory structures (for, let us say, a constant size Key On and a variable size System Exclusive).

Communicating

The second essential point is the communications achievement. The applications running under Midishare are not in charge of the communications physical execution but only deal with receiving and sending high-level events, for instance, complete Midi messages. In fact, MidiShare is internally in charge of translating events into corresponding Midi bytes sequences. Each application receives events into its own reception fifo. The application can consult this fifo anytime, extract and process the events waiting for treatment. Sending out events is easy as well. Stating the required output time and event to be transmitted will be sufficient.

Midi multiple ports can of course be used. On some softwares, Midi Channels are numbered from 0 to 31 so as to take into account the two ports which can be used on Macintosh computers. We thought preferable to separate these two informations. This way, each Midishare event has a channel number from 0 to 15 and a port number from 0 to 255; in the future, this will permit to deal with numerous Midi lines.

Time managing

It is obvious that a precise time control is essential for music. It is obvious too that according to the type of musical work carried out, different time representations are needed; which is why we chose the well-tried general solution consisting in an absolute representation of time in milliseconds over which other more «musical» representations can be built up by applications.

A 32 bits field defines the time for each MidiShare event. This field mentions the arrival time for the events received by an application or the required transmission time for events sent out by an application.

The algorithm used to manage the events scheduling is of course critical to the system’s performances. Several methods have been proposed (see for example [1,5] for a discussion on this point). In this particular case, we have used an algorithm developed at first by one of the authors for MidiLisp [2]. This very efficient algorithm ensures in all cases a bounded scheduling and dispatching time for treated events.

Tasks managing

Not only Midi messages are managed by MidiShare. Actually, an application having to remind a task to be done at a precise moment, can send to itself a particular type of message containing all the informations required for this task to be carried out. When arrived at maturity, this internal event is set by MidiShare in the application's reception fifo as if it were an external event. The application's reception routine will then accomplish the required task.

A second mechanism is implemented by MidiShare for tasks management. It is much more «procedural» and actually, very similar to Moxie’s «cause» [3,4]. It is in fact a delayed procedure call. For this to be done, MidiShare collects all the call parameters, together with the address of the routine to be called and generates a specific internal event. When this event has come to maturity, MidiShare restores the application context and actually deals with the call. It is to be noted that this call is made under interrupts, which means that some precautions have to be taken.

A third point related with tasks managing is the possibility for an application to define a reception alarm. It is a routine whose application gives MidiShare the address and which will be called each time new events are received (see example). Alarms therefore avoid an application having to consult periodically its reception fifo. Moreover, being called under interrupts, they allow for very efficient time responses.

Linking applications

As explained in the introduction, the reason for the creation of MidiShare was to permit several independant Midi applications to run simultaneously and to collaborate if necessary. This target may initially seem paradoxical. On the one hand, giving every application the impression to be the only user of the computer and consequently lessening the applications common resources. On the other hand, trying to connect these applications.

To isolate the applications was not a very difficult task. For instance, as regards incoming Midi messages, each application receives a different copy of them. Besides, every application possesses its own filters and reception fifo. There are no problems either to merge the various message transmissions, since they occur at the logical level too. Finally, as regards time, all applications refer to the same absolute clock in milliseconds, without having any influence on it.

How can we now link these applications ? Many users undoubtedly experienced Midi informations transfers, of musical sequences for instance, between two computers. We have taken over this idea and implemented inside MidiShare an internal real-time communication mechanism between applications.

Let us take a concrete example: two applications, the first being an echo generator, the second a sequencer. By default, these two applications are connected to the actual Midi I/O. However, it is quite possible to connect the output of the echo generator to the input of the sequencer. Each application is not aware of the connection change.

There is no constraint in the way to connect applications between themselves. In particular, it is possible for an application to have multiple sources and multiple destinations, or else to be connected to itself (for example a sequencer recording itself). Furthermore, these connections can be dynamically reconfigured while the applications are running.

Conclusion

After several months of using MidiShare, it clearly appears that the system brings a significant support in the creation and development of Midi applications and leads to a noticeable reduction in the development time and applications size. It also appears that the simultaneous use of several applications and the creation of links between them offer the user a genuine comfort. It actually becomes possible to use (and consequently to develop) small specific applications which would not be usable on their own, but reveal very interesting when connected with others. MidiShare therefore encourages a synergy between heterogeneous Midi applications.

MidiShare is now available on the Macintosh and the Atari, and new versions are planned for other computers. Several extensions of MidiShare are being developed in particular a network extension allowing connections and communications between several MidiShare stations via a Local Area Network.

Acknowledgements
We wish to thank everybody at Grame for their help and support in this project and also Dominique Fober for porting MidiShare on the Atari, and Georges Bloch for developping the MidiShare - C interface.

References
[1] Anderson D. and Kuivila R. 1986 : «Accurately Timed Generation of Discrete Musical Events». Computer Music Journal 10 - 3.

*[2] Boynton L., Lavoie P., Orlarey Y., Rueda C., and Wessel D. (1986) : «MIDI-LISP: A Lisp based Music Programming Environment for the Macintosh»
Proceeding of the ICMC 1986*

*[3] Collinge D. J. 1894 : «Moxie : A language for Computer Music Performance».
Proceedings of the ICMC 1984.*

*[4] Collinge D. J. and Scheidt D. J. 1896 : «Moxie for the Atari ST».
Proceedings of the ICMC 1986.*

*[5] Dannenberg R. B. 1896 : «A Real Time Scheduler/Dispatcher».
Proceedings of the ICMC 1986.*