MEDI-LISP

A LISP-Based Music Programming Environment for the Macintosh

Lee Bayman, Pierre Lavalle, Yann Orlandy, Camille Rodier, and David Weihs

1. ACT Informatics, 14, rue de la Montagne Ste-Geneviève, F-75205 Paris, France
2. INRIA, 8, Rue du Four, F-75634 Paris, France
3. IRCAM, 31, rue Saint-Maur, F-75006 Paris, France

Abstract

The evolution and current state of the MEDI-LISP music programming environment are described. Our design goal is a robust LISP, which we call MEDI-LISP, that is both extensible and re-sizable, that can be used for any type of sound manipulation. The current version of MEDI-LISP provides MEDI drivers, LISP-convertible music language MEDI event processing models, and schedulers, MEDI-based parts of the PARAFE - an object-oriented programming toolkit for real-time musical processes and graphic control panels (as implemented with Boyos et al. in the volume), and evolving class library of musical and graphical objects. MEDI data conversion and printing facilities, and LISP implementations of musical data-base manipulation tools based on M-LOGO (an object-oriented framework for music creation and organization). To make LISP usable in a real-time control language, direct control over memory allocation and deallocation is essential to prevent garbage collection. MEDI-LISP is implemented in Steel, a large extension to Common LISP, and is available to the public on the software distribution service. MEDI-LISP is implemented in a real-time and interactive style and is accompanied by documentation.

INTRODUCTION:

We are excited by this development because we are thought to exist individually, secondarily and still in a phase before the power of a professional LISP environment doing real-time processes can be (in fact, it is) connected to power symbols and processing and private sensing-controlled devices.

MEDI-LISP Environment:

MEDI-LISP is a software environment that provides LISP (Schaffer et al. in 1980) with the facilities to control and manipulate all types of musical manipulation. MEDI-LISP consists of a stand-alone set of LISP-extensible MEDI drivers for the Macintosh and a music-oriented environment and software toolkit that supports the manipulation of musical objects.

This environment under version 1.2 of LISP, a rich and efficient LISP developed at INRIA (Institut National de Recherche en Informatique et Automatique, Paris, Version 1.2 of LISP approaches the Common Lisp standard (Bolz, 1984) in that it is a rich and powerful language that can be used for real-time programming and object-oriented programming. The environment includes both an interpreter and a complex as well as complex graphics to the Macintosh ROM and Macintosh system.

A BIT OF HISTORY:

Software available at this point gives a brief overview of the history of this development.

In October of 1985, Apple, France kindly encouraged IRCAM to develop several software for the Macintosh LISP environment that would be used for its Macintosh laboratory. In an existing computer, we were initially decided by the fact of the high level language software development tool and by the fact that these were not widely available MEDI interface for the machine. IRCAM, LISP and LISP are the same language. Understandable, software is not yet called X, IRCAM went not entirely enthusiastic about assembly language programming or that matter an Apple in PASCAL Mac development environment. By December of 1984, Apple began to pick up when we learned that a new version of LISP that was being passed to the Mac. We obtained early versions of this language development from ACT Informatics, the software firm responsible for the port to the Macintosh.

The initial programming environments with LISP on the Mac were so exciting that one of the authors (Weihs) decided to test LISP in the principal programming language in his newly established Company Mac course at the Conservatoire National Superieur de Musique de Paris, where a Macintosh MEDI model went under development. The LISP environment, or the Mac and its conservation is sufficient to the combination of experiences with IRCAM has shown the LISP well adapted to musical applications.
and that it is an easy language for computer novice music teachers to learn. This we believe is due to LISP's symbol manipulation power, its orientation towards rapid interactive programming, and its very simple and uniform syntax that applies to both procedures and data.

In January of 1985, Camilo Rueda, with the encouragement of Xavier Rodet and Pierre Louvat, undertook the port of the FORMES environment (Couste & Rodet, 1984) to the Macintosh. FORMES is a large program written in LISP, and the fact that the port was successful and produced surprisingly efficient performance was both exciting and encouraging.

FORMES was producing convincing graphic representations of conversations on the screen of the Macintosh, but by as late as May 1985 there was still no MIDI on the Mac. Furthermore, with no clear decision on what the commercial world was set to do to decide which way to go with the MIDI interface and its drivers for the Mac. There were two possible options: an intelligent interface based on the Roland M16-AI, which reconnected the Yamaha YM 1233 MIDI interface chip, or with a simple interface that directly uses the serial ports on the Mac. While the development of a programmable smart interface that involves the Macintosh from the physical and time critical MIDI data handling tasks is being properly controlled, it is not important to mention, the fact that simple interfaces (like those built by CRONE/SYSTEM) have become widely available for the Mac. Another problem was the ability of the MIDIfied environment to import in the Macintosh data. In early June of 1985, after receiving a prototype of a simple MIDI interface from Automatique Processes Inc., Rueda completed a prototype interface, LISP, and the MIDIfied world, opening the possibility for real-time control of musical processes.

With the MIDI link established we were able to write real-time compositional algorithms in FORMES or directly in LISP, that were able to communicate with the rather demanding effects of a LISP graphics engine. A typical LISP package, connected to the serial port and running in, is the early implementation, the musical processor, that was triggered by events, resulting in line segmenting or sustained note. While we were aware of the problems we would have in not very concerning parallel problems, we did not realize that it would be so difficult to obtain memory allocation in our programs in such a way as to avoid the proliferation of core cells and the subsequent provision of potential bottlenecks. Real-time computing the production of core cells in every low-level function essential to real-time performance is a serious problem link with results of whatever.
MIDI LISP is currently in a state of continuing development. Many of its features have been designed and implemented independently and comprehensively, leading to an evolving interface. We now describe some of its characteristics.

MIDI DRIVERS:

The drivers for the Macintosh serial port service meet the standard commercially available MIDI interface—incorporating these produced by OCECIDE Systems and Soundbehe Music Systems. These drivers service both of the Macintosh serial ports simultaneously, providing two physical MIDI connections that both function with simultaneous input and output. Some interdriver drivers, designed and implemented in LISP, synthesize a common data stream that is internally available directly to the application.

MIDI EVENT SCHEDULING:

Event scheduling is the most critical issue to be concerned with. With current drivers and pre-processor major surgery to the code, these new events are added to the process, with scheduling now occurring in a perfectly precise, precise time, and does not depend on the duration of the execution of a piece of code in the program.

In our current implementation, there are two user-selectable approaches to scheduling. First, in a MIDI event processing discipline based on the MIDI events stored in the MIDI event processing layer design and implemented by Allen Einstein. When this assembly language module is enabled, a MIDI event event can be sent from LISP towards the drivers, but not vice versa. A delay can be inserted into the MIDI event output, and the event is stored in a queue for an event to be scheduled when it is activated.

OBJECT-ORIENTED SCHEDULING:

Let's start with a common example to demonstrate scheduling in the MIDI LISP environment to illustrate how LISP and MIDI data fit together with an object-oriented scheduling discipline. Consider the following message to the object:

```
MIDI-200: delay-clip time %s <delay> <start-clip>...
```

where \%s is a value in the universe representing the appropriate MIDI code of 90 (note).

Using the above example, we write a simple LISP function to play a note after a specified delay with a specified duration. In the following functions we have subjected out again any state changes and are left with concern about note length:

```
(defun play-note-time (note (start-clip) (duration))
  (delay-clip time %s <delay> <start-clip> <duration>)
)
```

where %s is a value in the universe representing the appropriate MIDI code of 90 (note).

Please note that the velocity of 0 is equivalent to a note of 0.
Also note that the code itself allows time to accrue within idle states that do not typically transform states. A counter time interval, or "scheduler-advance," is introduced between the time specified in the programmed real time. If no scheduler-advance is longer than the maximum execution time of the processes involved in an event, it will return at the specified time.

**AN EVOLVING CLASS LIBRARY:**

Using PREFORM in the MIDI-LISP environment an evolving library of objects has been developed. Synthesizer programs, and keying programs are for DIZA, P5C, P5A, MP6-90, MP7-90, P5D-90, MP8-90, P5A-90, and EVER, and recently in the Lassi PC20 are already in place and objects for wider music synthesizer processors like the DIZA 3000, amplifier and MP7 420 MIDI-equipped monos are in development. Other useful objects include a sequence class that includes methods for playing and recording sequences of MIDI events. An inheritance mechanism is provided for the creation of sub-classes of these objects (Koskimaki et al., 1986).

**SCORE REPRESENTATION TOOLS:**

Through MIDI-LISP does not provide for the full graphic representations of commercial music notation as one expects in a professional music editing and printing system, we intend to provide tools that will enable the generation of data files formatted appropriately for the available printing programs.

We feel that the traditional music notation is an important part of a music-composition environment. Some composers at IRCAM, CCMX, and University of Massachusetts developed some tools for developing traditional music graphics adequate for their compositional purposes. We do not think it easy to develop an environment that really does a good job in offering music notation oriented composition with the editing and printing of professional quality scores — without the rather frustrating link between Mark of the Unikate! Performance, a useful notation, and 18 bit printing program Professional Composer and the relation between Sibelius's Hand-Drawn musical notation and a more professional data-based graphic implementation. We feel that MIDI-LISP could be a very useful tool for the music composition. The next section will describe how MIDI-LISP could be put to use in the musical composition as its direct-connections between the worlds of musical notation and the worlds of professional data. One of the most interesting developments, towards a musically viable relation between graphic notation and professional data is the Autoscoring Transcription Project at Stanford's CEAMAS (March 1983). It depends on a highly interactive LISP environment.

MIDI-LISP does provide a multithread graphic representation of MIDI events and user with Mac style editing capabilities, inspired by MIDI-LOG (Orlare, 1985).

**CONCLUSION:**

MIDI-LISP is still under development but we feel that we have found effective solutions to some of the critical problems of programming in LISP for real-time. Garbage collection has been eliminated as a problem in our basic routines and it is possible in LISP to write very efficient garbage free code. Elements of PREFORM have proven useful for writing communicating parallel musical processors and control panel interfaces.

**ACKNOWLEDGEMENTS:**

We would like to thank Marylee, Punnow, Dolph, Xavier Rodet, Yves Foudrot, Jacques Dubro, Miller Packer, David Reznic, Jean-Baptiste Barjou, Jean Marie Hojo, and those at Apple France, ALC, ACT, MCEM, Stanford GRAME, IRCAM, and Yamaha who have aided in the development.

**REFERENCES:**


