The CMU MIDI Toolkit[1]

Roger B. Dannenberg

Computer Science Department and
Center for Art and Technology
Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.

Abstract

The CMU MIDI Toolkit is a collection of programs for experimental computer music education, composition, performance, and research. The programs are intended to allow low-cost commercial synthesizers to be used in experimental and educational applications. The CMU MIDI Toolkit features a text-based score language and translator, a real-time programming environment, and it supports arbitrary tuning and rhythm.

1. Introduction and Overview

The *CMU MIDI Toolkit* (CMT) is a software package designed for experimental computer music education, composition, performance, and research. CMT includes a compiler for a simple, text-based music language, and software for recording, playing, and computing MIDI data in real time. CMT has three major attractions: the flexibility of an open-ended design, the availability of source code, and low system cost. What does CMT do? The major components and their functions are described below:

Adagio is a language and an associated compiler. In Adagio, a note is represented by a line of text that specifies attributes such as pitch, duration, and loudness. Adagio is quite flexible and is compatible with several different ways of thinking about scores. For example, ‘‘Q’’ stands for a quarter note, but duration can also be indicated by ‘‘U87’’, which means 0.87 seconds. Adagio also supports arbitrary tuning systems.

Transcribe reads input from a MIDI keyboard or other MIDI controller and produces an Adagio score. The score can be played back using Adagio, or it can be edited with a text editor to make alterations. Since Transcribe output is at least somewhat readable by people, it can also be used for performance analysis.

[1]Published as: Dannenberg, ‘‘The CMU MIDI Tookit,’’ in ***Proceedings of the 1996 International Computer Music Conference***, International Computer Music Association, (October 1996), pp 53-56.

Record is a combination of Adagio and Transcribe. It plays an existing Adagio score while transcribing a performance. Thus, complex compositions can be created one part at a time as if using a multi-track tape recorder.

DXGet and **DXPut** are programs for recording and replaying MIDI system exclusive messages. These programs are typically used to save and restore synthesis parameters for a digital synthesizer.

Moxc is a real-time programming environment that is ideal for writing interactive music programs. Moxc is an extension of the C programming language and is based on Douglas Collinge's *Moxie* language.

Also provided are routines (in C) that allow direct production of MIDI output. Other routines are available to read MIDI data from a circular input buffer and to get the current time with 0.01 second resolution.

Required hardware and software. CMT runs on an IBM-XT or IBM-AT with a Roland MPU-401 MIDI interface and IBM-PC interface adapter. You should also have a screen-based text editor. To use Moxc, you will need a Lattice C compiler.

Other details. CMT is distributed by the CMU Center for Art and Technology, Carnegie-Mellon University, Pittsburgh, PA, 15213. We hope that users will contribute new software to the system and enhance the existing software in various ways. We will encourage this by helping to integrate and document new software and by distributing software updates to CMT users.

Below, we will provide more specific descriptions of selected components of CMT, followed by some general comments.

2. Adagio

Adagio is a small compiler for note-oriented scores. It has its roots in the score languages of Music V and related computer music synthesis programs, but Adagio is easier to use and more portable. Adagio also has the capability of expressing (sampled) continuous functions although work is needed to make this feature more useful.

A note is described in Adagio by a set of attributes, and any attribute not specified is ''inherited'' from the previous note. Attributes may appear in any order and must be separated by one or more blanks. An attribute may not contain any blanks. The attributes are: time, pitch, loudness, voice number, duration, and timbre.

Adagio has been used to program a variety of hardware and software synthesizers, and the Adagio compiler can be easily adapted to new environments [2]. Although not originally intended for MIDI, Adagio works quite well as a representation for MIDI scores. The MIDI version of Adagio currently uses the timbre attribute to select a MIDI ''program'' (synthesizer preset). Adagio has been extended to allow MIDI controller data such as modulation wheels, pitch bend, and volume.

3. Nonstandard Tunings

Tuning in MIDI is normally twelve-tone equal temperment. MIDI has no provisions to change this except by using the pitch bend control. In general, a different setting of pitch bend is needed for each pitch in a scale. Needless to say, this can be very tedious to program explicitly; however CMT has a way to automate the use of pitch bend to obtain the desired scale. The tuning mechanism in CMT is quite simple: whenever a program (Adagio, Record, Transcribe, Moxc, etc.) goes to play a note, the note's pitch is used as an index into a table of (pitch, pitch bend) pairs. The pitch bend is sent, followed immediately by the pitch from the table. Using the table, it is possible to translate every specified pitch into an arbitrary pitch and pitch bend.

Tuning, transposition, and pitch permutation are all just special cases of this general scheme. An interesting exercise is to map pitches according to some permutation (say, C to F#, C# to F, etc.) and to play a fugue with Adagio. Whenever the fugue theme is repeated at the octave, the permuted theme is also repeated, and rhythms are preserved throughout. In other respects, the pitch material is radically altered.

The use of this tuning mechanism is completely optional. If no tuning is specified, then notes are not translated and no pitch bend commands are sent.

4. Moxc

Moxc is based on the language MOXIE [1]. The best way to describe Moxc is to present a program that illustrates its features. The following program plays a sequence of notes with diminishing velocities to simulate an echo. The sequence is triggered by a pressing a synthesizer key and arbitrarily many sequences can be started by pressing multiple keys.

```
#define delay 15

echo(chan,pitch,vel) {
    vel = vel - 5;
    if (vel > 0) {
        midi_note(chan,pitch,vel);
        cause(delay,echo,chan,pitch,vel);
    } else midi_note(chan,pitch,0);
}

keydown(chan,pitch,vel) {
    cause(delay,echo,chan,pitch,vel);
}

mainscore() {
    cause(1000,mainscore);
}
```

This program is executed in an environment that continually polls for input from either the computer keyboard or from the MIDI interface. When input is found, the environment makes a call to a routine specified by the user. One of these routines is *keydown*, which is defined above. The *keydown* routine schedules the *echo* routine to run after a short delay by calling *cause*.

The *cause* routine is the heart of MOXC. Its first argument is a delay (in hundredths of seconds) and its second argument is the name of a routine. The *cause* routine schedules a call to the specified routine after the given delay. Any other parameters to *cause* are saved and passed to the specified routine when it is called.

Thus, the *echo* routine will be called 0.15 seconds after *keydown*. The *echo* routine begins by decrementing its *velocity* parameter. If the velocity parameter is still greater than zero, *echo* plays the given note and uses *cause* to schedule another call to *echo*. This will decrease the velocity further, play another note, and schedule yet another call. This process will repeat until the velocity goes to zero or below, at which time *echo* sends a note-off command and terminates.

Moxc is striking in its simplicity; parallelism is achieved with very little effort on the programmer's part, MOXC does not require any extra compilation steps beyond those of ordinary C programs, and the system is capable of running many concurrent activities on a personal computer.

5. Interface design issues

A few words about the overall design of this interface are in order. To begin with, CMT is neither a complete interface to the MPU-401 nor to MIDI. Instead, CMT is an attempt at providing the intended community of users with a rational interface that supports experimental, real-time computer music functions. One of the reasons CMT comes with source code is so that if you disagree with these design decisions, you are free to modify or extend the system to meet your requirements.

The main areas in which CMT deviates from the ``conventional'' are the absence of ``tracks'', the way in which time is handled, pitch specification, and the lack of external synchronization. Tracks are a concept implemented in the MPU-401 whereby several sequences of MIDI data can be merged in real-time. In CMT, the Adagio compiler sorts its data, so tracks are not needed to play multiple sequences together. For example, to play two Adagio scores simultaneously, one can normally just concatenate the files together and run Adagio on the new file.

Timing in CMT is probably the most radical departure from MIDI. Whereas MIDI sequencers normally tend to talk about time in terms of beats, CMT measures time in units of 0.01 seconds. This is roughly the smallest rhythmic time deviation we can perceive. The rationale behind this decision is that not all music is measured in beats, and some music has several tempi going simultaneously. If everything is converted to time in seconds, then one can freely combine scores with different tempi and meters as well as scores with timing notated directly in seconds. Another timing issue is that the MPU-401 was designed to allow the host computer to send data in advance of its actual use. This is not very suitable for interactive real-time programs in which one normally wants output to occur immediately after data is sent to the MPU-401.

Pitch in CMT is based on earlier computer music systems in which middle C is represented by the number 48. Therefore, CMT pitch numbers are 12 less than the corresponding MIDI pitch numbers. CMT also allows users to redefine the interpretation of pitch numbers as described in Section 3.

Finally, CMT at present has no means for external synchronization and cannot now be used with other sequencers or drum machines to achieve a synchronized performance. This is partly a consequence of the fact that CMT does not measure time in beats, while sequencers synchronize by sending MIDI messages to mark beats and their subdivisions.

6. Applications

CMT is quite practical for realizing computer music compositions. The composition ‘‘Jimmy Durante Boulevard,’’ presented at this conference, is implemented using CMT. CMT is also useful as a vehicle for introducing programming to musicians [3]. The CMT manual documents a subset of C that is adequate for simple compositional work, and the system is used in computer music courses at CMU.

Another application of CMT is to supplement an existing computer music installation. Adagio, Transcribe, and Record can be used to prepare and fine tune scores before subjecting them to a relatively slow realization using software synthesis. Although software synthesis is not a part of CMT, it is a simple matter to modify the Adagio compiler to output scores in your favorite language. Alternatively, you can modify your favorite composing program to output Adagio scores and use CMT to play them on MIDI synthesizers. Thus, CMT can help bridge the gap between MIDI equipment and a software synthesis environment.

7. Conclusions

CMT was implemented because no other system offered the sort of flexibility and portability we needed. In particular, CMT differs from existing MIDI software in the following ways:

1. CMT attempts to avoid dependencies on MIDI; CMT has been used with non-MIDI synthesizers and software synthesis.
2. CMT offers a flexible approach to time and pitch, supporting polyrhythms, multiple tempi, and arbitrary tunings.
3. CMT is open-ended. The use of readable and editable text-files to encode MIDI data, and the availability of source code for CMT programs makes the system very flexible. Adagio has become a standard medium of exchange between many programs at CMU because of its simplicity, and new applications are relatively easy to write because of Moxc.

Currently, CMT runs on IBM-PC, IBM-AT, and compatible computers using a Roland MPU-401 or compatible interface. The system is written in Lattice C. Work is in progress to port CMT for use with other compilers and other machines, and the author invites anyone interested to join in this effort.

8. Acknowledgments

Many people have contributed to CMT which grew out of earlier computer music systems used in the CMU Computer and Electronic Music Studio. Dean Rubine wrote the first MPU-401 interface routines with help from Dale Amon. Joe Newcomer ported this first version to his IBM-AT and helped clear up many of our interrupt problems. Aaron Wohl wrote routines to use the PC timer and made many good suggestions for improvements. Michael Cressman wrote the original tuning software. Most of the development work was done on the IBM-XT in the CMU Computer and Electronic Music Studio. This machine was part of a grant from IBM.

I also wish to acknowledge support from CMU through the Music Department, the Computer Science Department, the Center for the Design of Educational Computing, and especially the Center for Art and Technology.

References

[1] Collinge, D. J.
MOXIE: A Language for Computer Music Performance.
In W. Buxton (editor), *Proceedings of the International Computer Music Conference 1984*, pages 217-220. International Computer Music Association, 1985.

[2] Dannenberg, Roger B., Paul McAvinney, and Marilyn T. Thomas.
Carnegie-Mellon University Studio Report.
In W. Buxton (editor), *Proceedings of the International Computer Music Conference 1984*, pages 281-286. International Computer Music Association, 1984.

[3] Dannenberg, F. K., R. B. Dannenberg, and P. Miller.
Teaching Programming to Musicians.
In D. Mansfield (editor), *Proceedings Fourth Symposium on Small Computers in the Arts*, pages 114-122. IEEE Computer Society, Washington, D.C., October, 1984.