WPI-CS-TR-93-3 July 1993

Music Related Computer Science MQP's

by

Roy S. Rubinstein

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Music Related Computer Science MQP's

Roy S. Rubinstein
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609
roy@cs.wpi.edu

July 1993

Abstract

One important way to teach Computer Science (or almost any discipline) is by getting students involved in projects that interest them. One such area of projects involves the use of computers with music. This paper presents some experiences I have had advising music related computer projects. These experiences show that students are very enthusiastic about music projects, enjoy working hard at them and learn a lot in the process.

This paper also serves as an introduction to MIDI, the standard interface for electronic music. It provides enough background so that one may get started advising or doing MIDI projects.

1 Introduction

The use of projects in teaching Computer Science is standard practice, but it is often difficult to find appropriate projects. The project must not only be of the proper difficulty level for learning, but should also be of interest to the students. One area of projects that greatly interests many students is the use of computers with music. This paper presents some experiences I have had advising such projects.

As in many areas, computers are becoming increasingly prevalent in music composition, performance and production. Many types of music and sound equipment may be controlled by computers, including some instruments themselves. In particular, synthesizers and samplers, which produce sounds, may be controlled remotely by a keyboard or a computer. The Musical Instrument Digital Interface (MIDI) is the standard through which the communication is done. Of course specialized software and hardware is needed to perform this communication with the MIDI standard, and many such projects are possible in their design and implementation.

This paper is organized as follows. First is an introduction to the Major Qualifying Projects at Worcester Polytechnic Institute, the program under which these projects were and are being pursued. This is followed by an introduction to MIDI, the electronic music communication standard. Today most electronic music uses the MIDI standard, and as such so do all these projects to some degree.

Next are descriptions of computer music projects that have already been completed, along with comments concerning their educational value. The objectives and functionality of the projects are described here, though the details of the implementations are omitted. The emphasis here is on what the projects do, not how they do it. Further information on the projects may be obtained from the project reports.

This is followed by descriptions of current projects, scheduled future projects, and potential future projects. Problems encountered with the projects are also discussed.

2 Major Qualifying Projects

At Worcester Polytechnic Institute part of the undergraduate degree requirements is the completion of a Major Qualifying Project (MQP), which is a large project in a student's major that is equal in credit (and work) to at least three courses. A report on the project is required at its conclusion.

The types of projects vary greatly, and a topic may be anything whatsoever as long as a faculty member in the appropriate department approves and agrees to advise it. While an MQP may be completely theoretical, this is very rare, and large programming projects are standard in the Computer Science Department. Projects may be done by individual students or in groups of various sizes, with the size of the project appropriate to the size of the group.

The purposes of the projects vary somewhat with the project, but in general should include a reasonable subset of the following list.

- to apply already learned Computer Science knowledge and techniques
- to gain new knowledge in Computer Science

- to explore an area in greater depth than time usually permits in classes
- to gain experience in learning new material outside of a classroom
- to learn about a non-Computer Science area
- to explore the relation between Computer Science and other disciplines
- to gain experience in a large scale design and development project
- to gain experience in writing a project report
- to produce a useful, original, high-quality product

Different advisors prioritize these items in different ways, often depending on the individual projects, though using current and gaining new knowledge is virtually always expected. While a nice end product is desirable, it is by no means the only goal. Having students design and develop a product that does not compare favorably to a similar commercial product (that may have been years in development) may be quite acceptable. In fact, often while "reinventing the wheel," a new, original technique may be discovered.

There are many projects in computer music that fulfill many of the above goals very well. Some of these will be described later in this paper.

3 Introduction to MIDI

The Musical Instrument Digital Interface (MIDI) standard [MID90] is today's standard for communication between digital musical instruments. It includes standards for real-time communication between instruments as well as file formats for storing musical information. Almost all high quality electronic musical instruments currently being manufactured use the MIDI standard.

3.1 Real-time communication

In its simplest form, MIDI can be used to allow a keyboard to control a separate sound module, which could then be connected to an amplifier and speakers. This set-up is shown in Figure 1. Sound modules typically fall into the categories of *synthesizers*, *samplers* or a combination of the two. Often a keyboard and sound module are combined into a single unit, but here they will be considered to be logically separate.

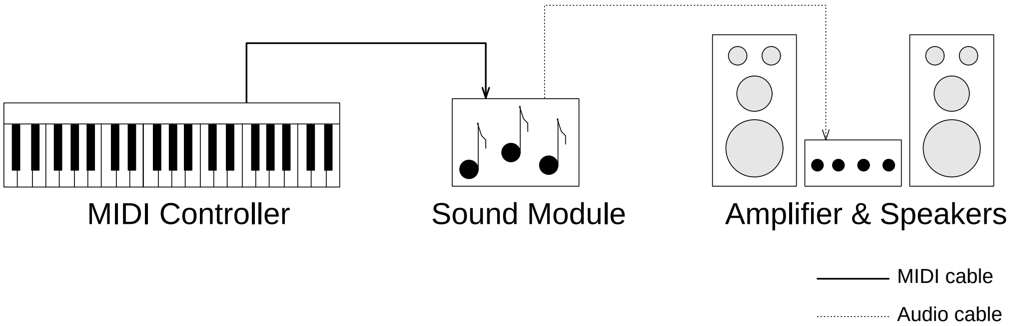

Figure 1: Simple MIDI set-up.

Using MIDI allows easy replacement of one component independently of another. In addition, with the appropriate connections (and possibly other equipment) multiple sound modules may be controlled from a single keyboard, and multiple keyboards may share a single sound module.

Most sound modules are capable of producing a number of different sounds, and some sound modules and/or keyboard controllers may be set up to vary the sound depending on the pitch (i.e. what note was played), the velocity (how hard the note was hit), the channel on which the note was sent (which could vary depending on which keyboard the note was played), or other parameters.

There are a number of MIDI messages that can be sent in real-time communication. The two primary messages are NOTE ON and NOTE OFF. The NOTE ON message indicates that a note is to be played, and includes information about what note, what velocity, and which channel. The NOTE OFF message is used to stop playing a note previously started by a NOTE ON message, and includes information about what note to turn off on which channel. A note can also be turned off by a NOTE ON message with velocity 0.

Other MIDI messages have the ability to change the instrument sound associated with a channel (program change), to bend pitches, to modulate the sound (such as changing vibrato depth), and to alter the volume of a note already being played.

The MIDI standard includes both the formats of the messages and the hardware specifications for the communication.

If a computer with a MIDI interface and the appropriate software is configured into the MIDI set-up (for example as in Figure 2), the capabilities are greatly increased. When notes are played on the keyboard, in addition to sending the messages to the sound module to be played, the computer can save the timing and message information. This can then be resent to the sound module to play the notes again, without the need to play on the keyboard. The note sequence may also be edited to correct mistakes, add notes,

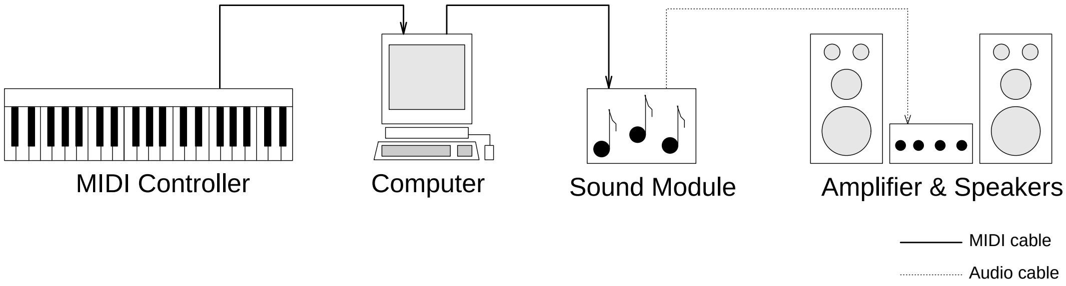

Figure 2: Simple MIDI set-up with a computer.

change the tempo, etc. One may also play back a stored sequence while playing a new sequence on the keyboard, possibly with a different instrument sound, which can then also be stored. This process, known as *sequencing*, allows a single person to orchestrate an entire score.

3.2 MIDI files

The MIDI standard also includes specifications for file storage. This allows a sequence stored using one sequencing program to be used by another, even by one on a different type of computer. It also gives other types of programs access to the same information. An example of a program that may need such access is a scoring program that takes a MIDI file and produces sheet music. Or a multimedia application may play any specified MIDI file. MIDI files are discussed in greater detail in the section about PRIMA.

Another type of file used with MIDI is one that allows storage of groups of instrument parameters, known as *patches*. This type of file is not of a standard format and is not usually referred to as a MIDI file. This is discussed below in the section about a patch editor/librarian.

3.3 Other MIDI applications

The MIDI standard is used in many applications, sometimes with extensions to the standard. There are many audio effects devices (such as reverb, digital delay, chorus, and combinations of these) that can be controlled via MIDI. There are even mixers and equalizers that can be controlled via MIDI. This allows easily synchronized control of all sound. A MIDI standard for lighting control is also in use.

4 Completed Projects

This past year I began offering MQP's concerning MIDI, and four have been completed. This section discusses these projects and what computer science and other knowledge was used and learned in doing them. In the discussion of the projects, more information about MIDI is provided.

4.1 PRIMA: Translation of MIDI files to sheet music

The PRIMA project (as the students involved named it) was the development of a system for creating sheet music from a MIDI file. After generating its "best guess" at what the sheet music should look like, it allows adjustment via an X Windows graphical user interface. The output is a file that is run through TEX with MusicTEX. (MusicTEX is a package of macros and fonts for producing music with TEX. It is very unstructured and detailed, and one does not want to use it directly.) By using standard interfaces such as X Windows and TEX, this can be quite portable.

Although there are many commercial programs available to print sheet music from standard MIDI files, sometimes incorporated into sequencer programs, there are none that I am aware of that are designed for workstations running in an X Windows environment. With workstations becoming more prevalent, powerful and affordable, particularly in academia, this is a very useful tool.

A MIDI file contains all the timing and note information needed to properly play the music, though it may not explicitly contain the necessary information for producing sheet music. For example, there is information that a given note is started at a certain time (a NOTE ON message) and later information that that note is turned on (NOTE OFF), without explicitly giving the duration of the note. This must be calculated and, using other information about how the timing relates to quarter notes, determine what type of note (quarter note, half note, etc.) this translates into. If this note continues into the next measure (which, of course must be determined – a MIDI file does not give timing information in terms of measures but in terms of time units known as *ticks* or *delta times* from the beginning), further processing must be done.

It is also not always clear from the MIDI file just where on the staff a note is to be placed. The pitch is represented as a number, so, for example, note 58 can be interpreted as either B-flat or A-sharp (the one nearest to middle C). It is up to the program interpreting the file to make this determination. Also, if the the same note occurs twice in the same measure, the second would not typically need the accidental.

There are actually more than one type of MIDI file. A type 0 MIDI file contains a single multi-channel track, while a type 1 file allows more than one simultaneous track of a sequence. Type 2 files, not as commonly used, allow one or more sequentially independent single-track patterns. Specification of time signature, key signature, and other miscellaneous information is allowed but optional. PRIMA will accept type 0 and type 1 files, guessing at the time and key signatures if they are not provided.

The basic action of PRIMA is as follows. When begun, a window is drawn with two staves and several buttons to control the program. The first action is to read in a MIDI file, which involves parsing the MIDI file, calculating and storing information in the process. The output of this phase in an intermediate file (whose format was designed as part of the project) that contains information about notes (in terms of pitch, note duration, accidentals, etc.), MIDI-clock equivalents, page layout, etc. Once this intermediate file has been generated, the MIDI file is no longer used.

Next, the information from the intermediate file is used to draw several measures of the score in the window. This may be scrolled about to access different parts of the score. Changes may be made here, including moving notes, adding accidentals, changing key and time signatures, etc. When changes are saved, it is in the intermediate file format, with the original MIDI file unchanged.

Another action is to translate and display the music. This will translate the intermediate file into MusicTEX format, run TEX on the file, and display the output with xdvi (a standard utility for displaying a TEX .dvi file on an X Window display). If further changes are needed, they can be made and the process repeated. When a final version is obtained, the .dvi file can be printed by the usual means for the computer system configuration. The intermediate file can also be saved, so that one need not go back to the original MIDI file to make additional changes at a later time.

The PRIMA project turned out to be an extremely large undertaking. It involved learning the MIDI file format, some music theory (to break the input into different "tracks" to put on separate staves, allow proper beaming, etc.), some music notation, MusicTEX and TEX, X Windows, and some process communication (to run TEX and xdvi while running the other programs). Many complex data structures and algorithms were used and developed to efficiently implement this.

In fact, it turned out to be too large for the three students involved. As a result much of the functionality was trimmed, but the basic structure was still built. The graphical interface editing capability was not fully implemented, and the only changes that can be made there are movements of notes, and this still has some problems. Additionally, the user interface is not as comfortable and intuitive as it should be. As a result, the program as it now stands is far less functional than many commercial products (and equivalent to a public domain MIDI to MusicTEX program).

But the overall structure – setting up the X Windows interface, translating to the intermediate file, translating to MusicTEX, and interactively running TEX and xdvi – works. This is an ideal situation for a future project, which can fix the bugs and increase the functionality.

Although the final product was not completely successful in terms of the original plans, the students learned a great deal and accomplished much. At the same time, they enjoyed themselves (well, most of the time). As such, I would say that the project, in terms of the experience it is meant to provide, was undoubtedly a success.

4.2 Sheet music and MIDI file format for an expert system

This project, completed by one student, produced a pair of programs to translate from the nonstandard musical notation output by an expert system to sheet music (in the form of a MusicTEX file) and to a type 0 MIDI file. These programs work completely and properly as specified.

An instructor in our department for her Masters thesis wrote an expert system that produces variations in the style of Telemann [Mer91]. The output (as well as the input) for this system is a unique system that consists of triples, each representing a chord, a rhythm and a note. Without an enormous amount of practice, this is virtually unreadable. The programs produced by this project allow the expert system format to be easily read and listened to.

This project involved learning the format of the expert system output (which was not fully specified and, it turned out, had ambiguities), MusicTEX and MIDI files. Parsing and translation techniques were used to read the input and convert it to the appropriate output formats. Additionally, the student doing this project delved deeply into musical notation, and the sheet music produced is high quality, including beaming and proper stem direction.

The report included recommendations for extensions to the expert system notation to allow better processing of output. A number of these will most likely be adopted, as a project due to begin in the fall (not advised by me) will deal with the expert system.

4.3 MIDImapper: A multiport MIDI router

The MIDImapper is a hardware/software project completed by two students, one a Computer Science major and the other an Electrical and Computer Engineering major. The project was co-advised by Prof. William Michalson of the ECE Department, who was far more helpful with the hardware than I could have been. My discussion here will focus on the software and programming aspects of the project.

The MIDImapper is a user-programmable MIDI merger/splitter that takes two MIDI input streams and, in real-time, routes each message to either, both or neither of the two MIDI output streams. This device allows two MIDI controllers (such as keyboards) to share two sound modules. It can also be used for merging two separate streams (perhaps from two separate keyboards) into one (such as to share a single sound module or to simultaneously record the two streams with a single computer sequencing program). Many other uses are possible as well.

Because of the nature of the digital MIDI messages, even for the relatively uncomplicated merge operation, a simple "Y" connector can not be used. It could cause parts of incoming messages to be interspersed, and as such will not work correctly. A device such as the MIDImapper is needed that will store incoming messages from one input until the message from the other input has been completely output.

The routing of a particular message can be based on a number of factors, including channel, pitch and velocity. This allows, among other things, range splitting (so that different notes on a keyboard will play different instruments) and velocity sensitive mapping (so that a different instrument will sound if a note is hit hard). Transposition and channel number changing may also be done. Similar products are currently on the market, but this could be produced to be quite competitive. It was also designed so that it could be easily expanded to a greater number of input and output ports.

Two standard program settings are provided (straight, where messages from one input port go directly to one output port and messages from the other input port go directly to the other output port, and merge, where messages from both input ports go to both output ports), and battery-backed RAM is available for storing many user-programmed settings. The programming is done via a keypad and LCD display, which can also be used to view current programs. The LCD display also shows when MIDI messages are passing through.

This project involved designing and building everything, including the hardware as well as the software. In addition to learning the MIDI specifications (hardware and messages), the students had to program the system at a level quite different from what they were used to. Designing and developing software specific to the unique hardware architecture was a new experience for them. There was no operating system, and all the ports and memory locations had to be directly addressed. Timing was also critical.

The programming environment was also not one that most Computer Science students experience. Emulators were needed in the early stages, and the program (written in C) had to be compiled on one machine then loaded into ROM to be used in the MIDImapper.

The memory locations of various code (such as startup code) were critical, and there were numerous problems to overcome.

The project, both hardware and software, worked quite well. The students are planning to add additional functionality, not for credit, but for fun and to improve the MIDImapper.

4.4 RAGE MIDI guitar signal processor

The RAGE MIDI guitar signal processor was an ambitious project undertaken by one student to modify a digital guitar effects processor (analog input to analog output) to additionally provide a real-time MIDI output stream, allowing the guitar to be played with the sound of other instruments or sequenced. This involves polyphonic pitch detection, an area currently under research that is not considered to be fully explored or solved.

The student doing this project was a double major, Computer Science and Electrical and Computer Engineering, and as such had to do two MQP's. For his ECE MQP he designed and built, with another student, the RAGE (Rabid GEcko Electronics) digital guitar processor. For his CS MQP, he modified it to give MIDI output.

Even though most of the hardware was already built (a MIDI output port needed to be added, plus a MIDI input port was added for testing and future use), much work needed to be done. An in-depth study of algorithms to convert waveforms from the time domain to the frequency domain was done, and it was determined that Goetzel's algorithm was most appropriate (more so than standard Fast Fourier Transform algorithms). One major problem here is that a greater number of waveforms is needed to more accurately determine the pitch, and with lower frequencies this may take more time than acceptable.

Once the input is in the frequency domain, it must be determined what was played and at what volume. While this is not easy when only one note is played at a time, it is extremely difficult with multiple (up to six, with a standard guitar) notes. A harmonic of one note may be louder than the fundamental of another note, making it quite difficult to determine just what was played. Additionally, the relative amplitudes of a fundamental and its harmonics may be different if the note is played differently and may vary with time. A number of algorithms were proposed, but there was not sufficient time to try them all. Good quality polyphonic note detection was not achieved by the end of the project, though it worked well (and impressively) when only one note at a time was played. It should be noted that commercial MIDI guitar processors avoid the problem of polyphonic pitch detection by having six pickups, one per string, and doing monophonic pitch detection on each.

The programming was also quite different from what most Computer Science majors (or computer scientists anywhere) are used to. In addition to programming a standard microprocessor and the need to address everything without the assistance of an operating system, a digital signal processing (DSP) chip had to be programmed in its own unique way. In order to do this, all programming was done in assembly languages.

The student also set up his own development environment. He built a MIDI interface for his Macintosh computer, enabling him to edit and assemble his code on the Mac, then download it (through the MIDI interface) to RAM on the RAGE board for testing. When a debugged revision was done, it could be burned into ROM and plugged into the circuit board. The student intends to continue to work on this, not for credit, in order to add functionality (such as string bending) and to try different polyphonic pitch detection algorithms.

5 Current and Future Projects

5.1 A Windows MIDI sequencer

A MIDI sequencer on a computer is similar to a multitrack tape recorder but using MIDI messages instead of audio signals. In fact, usually there is a "control panel" that is designed to look like the controls of a tape recorder, with stop, play, record, fast forward, and rewind buttons. With such a sequencer one usually has the ability to play back any combination of the tracks already recorded and to record new tracks at the same time.

In addition, most sequencers have the ability to edit the sequences recorded to correct mistakes, adjust timing, copy passages, transpose, etc. Usually the editing is simple (adjusting one note at a time or a group of notes as a unit for copying or moving), but some allow algorithmic editing (such as defining a curve for velocity over a time interval). They usually also allow one to read in MIDI files and store sequences as MIDI files.

This project is to write a sequencer that runs on PC-compatibles under Microsoft Windows. There are already many sequencers available for PC-compatibles, some which run under Windows, but this one is to be easily extensible. If one wants to add new editing functionality or a new type of display, for example, it should be relatively easy to incorporate that new module into the system without the need to recompile the entire system. In fact, it should not even be necessary to have the source code to the rest of the system.

A basic sequencer is a rather large project, and depending on the functionality, this could be extremely big. A basic sequencer will need an extensive graphical user interface, including many displays, menus and buttons. It will need to access the MIDI interface and do precise timing. Complex data structures will also need to be developed to properly handle the information. Windows and MIDI (both file format and messages) need to be mastered.

The project is being done by one highly motivated student. (I tried to convince him it was too big to do on his own, but he was determined to do a Windows based sequencer.) Sketches have already been made as to what the screens will look like and what the functionality of the sequencer will be, and the implementation should be completed this fall.

I expect this to be another project to be extended by a future project group.

5.2 Computer-aided harmony

This project is being pursued by an individual student as a one-term independent study (as opposed to as an MQP, as the other projects discussed here are). The idea is to take as input a melody line perhaps with some other lines (such as a bass line) in the form of a MIDI file and to generate an additional harmony line. This project requires extensive musical knowledge as well as computer expertise.

The student started this and soon discovered the enormity of such a project. He also became overwhelmed with other coursework and took an incomplete on this. He is expected to finish it (or at least finish one term's worth of it) this fall.

5.3 A universal patch editor/librarian

Most sound modules can be programmed to produce different sounds, but the techniques of programming them vary greatly not only between different manufacturers, but between different models by the same manufacturer. There are, however, a few principles that almost all have in common. In general, a particular "instrument" is created for a particular MIDI channel so that all MIDI messages coming in on that channel affect that instrument. Multitimbral sound modules allow different instruments to be played at the same time by recognizing messages on more than one MIDI channel. The sound produced by an instrument depends on the programming of that instrument.

An instrument is programmed by setting a variety of parameters on the sound module. These typically include one or more waveforms to be used (either generated or sampled), sound envelopes for the waveforms, information about how they are to be combined, velocity sensitivity, aftertouch sensitivity, transposition, and many others. The parameter types and value ranges are completely dependent on the particular model of the sound module. Different modules may produce sounds in completely different ways, and new methods are constantly being introduced. Defining any kind of standard for this would not only be virtually impossible, but would limit technological advances. The set of parameters for a particular instrument is referred to as a *patch*, and the terms "instrument" and "patch" are often used interchangeably.

The method of programming most sound modules is by pressing a sequence of buttons to display the current parameters one at a time on a single line LCD display, with modification to a displayed parameter allowed. This can be quite cumbersome and slow. A patch editor is a computer program to allow editing of patches, usually with a nice graphical user interface displaying many, if not all, the parameters at once. While some parameters may be displayed with numbers in tabular format, some may be displayed in the form of graphs (useful for seeing an envelope, for example), meaningful text or other means, and may be changed by means of mouse clicks and drags, menus, keystrokes, etc. Patches may be downloaded from the sound module to the computer, edited, and then uploaded back to the sound module, all through the MIDI interface. This is done via MIDI system exclusive (SYS EX) messages, which are not standard, but are specific to a module manufacturer and model. Note that the simple set-up shown in Figure 2 will not suffice for this; a MIDI connection from the sound module back to the computer is needed.

Most sound modules have very limited patch storage space, and the manufacturer may sell expensive sound cards for additional, external storage. If one has a sound module connected to a computer through a MIDI interface, however, with the appropriate software the patches may be saved in files on the computer. This is where the librarian portion of a patch editor/librarian comes in. This allows patches to be organized and stored on the computer, either individually or in "banks". Most sound modules have the capability to have an entire bank of patches loaded at once, which is more efficient than loading each patch separately. A patch editor/librarian usually allows editing of banks as well as individual patches, so that a set of instruments may be easily loaded for a particular piece of music. There is no standard for the file format for patches or patch banks, though often if a software manufacturer produces a patch editor/librarian for the same sound module but for different computers, the files will be interchangeable.

When patch editor/librarians first came out, one needed a separate program for each sound module, even if the programs were to be run on the same computer. Today universal patch editor/librarians are becoming more common, where a single program is run with different configuration files for different sound modules. If a new sound module is added to one's system, it is only necessary to use a different configuration file with the same patch editor/librarian.

This project will be to design and implement a universal patch editor/librarian. This will include the design of the configuration file format so that it is as general as possible, plus complete documentation so that users may create their own configuration files. The project will begin this fall and will be done by three students. It will require knowledge of MIDI, sound modules (including their programming), data structures, files, and graphical user interfaces.

5.4 Potential projects

There are many other computer related music projects that can be done. As above, most of them have to do in some way with MIDI, though the MIDI portion may only be to have a standard I/O interface (as with the computer-aided harmony project). Potential projects include extensions to the aforementioned projects, including porting them to other computers. Integrating these and other projects into a single, cohesive unit would also be a worthwhile project.

Other potential projects include MIDI drum machines, ear training programs, waveform editors, and MIDI control systems. Projects such as computer-aided composition, musical analysis, and instrument modeling are possible as well, though they might quickly get to a level of research beyond the scope of undergraduate education.

6 Problems and Conclusions

The main problem I have had with advising music related projects is the lack of equipment available. While Computer Science departments often have enough general computing equipment, music equipment is rarely available. Two of the above projects dealt only with MIDI files, as no music equipment was needed for these (except for checking output). The other projects rely on the students' own equipment (along with my own personal equipment which I have lent out). As such, it not only limits which students can work on what types of projects, but to some extent it also constricts the breadth of their learning, as they are using equipment with which they may already have a great deal of familiarity.

Another problem is that, as a project is usually begun at the beginning of a student's senior year and sometimes during the junior year, the appropriate background courses may need to be taken concurrently with project work instead of before, particularly as some courses are only offered every other year. This, however, is a problem with virtually any kind of project, undergraduate or graduate.

In spite of the aforementioned problems, the use of music and MIDI in Computer

Science projects has been very successful. The students tend to be extremely enthusiastic about it, have fun doing it, and end up with useful products that they take pride in. And in the process, they learn a great deal.

References

[Cab93] Benjamin "Quincy" Cabell. Computer-aided harmony. Independent study report, 1993. Not yet written.

[CK93] C. Brian Candiloro and Jonathan Kemble. MIDImapper: A multiport MIDI router. Worcester Polytechnic Institute MQP Report RSR-9204, 1993.

[DMS94] Anthony Defusco, Robert Martino, and Daniel Steffann. A universal patch editor/librarian. Worcester Polytechnic Institute MQP Report RSR-9305, 1994. Not yet written.

[GSZ93] Carlos Gonzalez, Elizabeth Stewart, and Michael Zarozinski. PRIMA: Translation of MIDI files to sheet music. Worcester Polytechnic Institute MQP Report RSR-9203, 1993.

[Kin93] Brian King. RAGE MIDI guitar signal processor. Worcester Polytechnic Institute MQP Report RSR-9301, 1993.

[Mer91] Kathy Johnson Merck. An expert system to generate musical variations in the style of Telemann. Master's thesis, Rochester Institute of Technology, 1991.

[MID90] MIDI 1.0 detailed specification. The International MIDI Association, 1990.

[Sei93] Richard Seiffert. A Windows MIDI sequencer. Worcester Polytechnic Institute MQP Report RSR-9303, 1993. Not yet written.

[Szy93] Paul Szymkiewicz. Sheet music and MIDI file format for an expert system. Worcester Polytechnic Institute MQP Report RSR-9206, 1993.