Carnegie-Mellon University Studio Report

Roger B. Dannenberg
Paul McAvinney
Marilyn T. Thomas
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

Computer music activity is taking place in three separate facilities at Carnegie-Mellon University. The Computer Music Studio is is equiped with computer, recording, and synthesis equipment, and is used for teaching, composing, and the recording of concerts. The Solfege Lab is a cluster of personal computers, some with inexpensive synthesizers attached, for computer-assisted instruction in basic music skills. The Computer Music Laboratory is a research facility that supports several projects, including Arctic, a real-time control and composition language, and Vivace, a rule-based AI system for composition. Work is also being done in the area of real-time accompaniment and the development of a personal computer music system: the Musician's Workbench.

1. Introduction

In the summer of 1983, Carnegie-Mellon University constructed a Computer Music Studio for education, composition, and recording. At the same time, a laboratory was opened for computer music research. Already in place was a Solfege Lab for computer-assisted instruction. These three facilities represent a cooperative effort by members of the Music, Computer Science, and Computer and Electrical Engineering Departments. We will summarize current activities in these facilities and describe some of the work planned for the near future.

2. The Computer Music Studio

The Computer Music Studio is housed in the College of Fine Arts Building in a room adjacent to the Alumni Concert Hall, where most of the Music Department concerts and recitals are presented. This location enables the facility to serve three separate but interrelated functions: 1) a classroom for courses in electronic and computer music; 2) a studio for the composition and performance of new music using analog and digital equipment, sometimes in combination with live voice and/or acoustic instruments; and 3) a recording studio for performances in the concert hall, for out-of-studio recording work using studio equipment, and for dubbing tapes.

For recording purposes, the facility is equipped with a Scully half-inch quadraphonic tape machine, Teac quarter-inch quadraphonic recorder, Sony quarter-inch stereo machine, and two Technics cassette decks. An Audioarts twelve input, four-output mixer with four channels of DBX noise reduction, two stereo amplifiers, and four studio speakers complete the basic recording system. The studio is also equipped with a video monitoring system to enable remote recording of live performances in the adjacent concert hall.

As an educational facility, the Computer Music Studio is equipped with an ARP 2600 synthesizer used to demonstrate basic analog signal processes, and as one of the sound sources for the four compositional projects required of students taking Electronic and Computer Music I. This course is designed as an introduction to acoustics and to electronic and digital sound processes; the historic evolution of electro-acoustic music is also taught through outside readings and listening to musical examples. The focus of the course is on the students' own compositions, the best of which are included in two concerts of electronic and computer music, presented each year by the Music Department.

Additional sound sources in the studio are the Roland Compumusic System, the Soundchaser software, used with an Apple IIe computer, and the CMU digital

Dannenberg, McAvinney, and Thomas, "Carnegie-Mellon University Studio Report," in Proceedings of the International Computer Music Conference 1984, Paris, France, October 19-23, 1984. Ed. William Buxton. San Francisco: Computer Music Association, June 1985. pp. 281-286.

synthesis system, a low-cost system assembled from commercially available hardware and software written at Carnegie-Mellon University. The sound source for this system is a Mountain Hardware music synthesizer, which uses an Apple IIe computer to generate amplitude and frequency ramps. The system is controlled by IBM XT and a Sritek 68000 coprocessor running under the Xenix operating system. The system can be programmed in Adagio, a note-oriented symbolic score language developed at Carnegie-Mellon University, or in the C programming language, which offers greater flexibility but requires more programming ability. Both languages are taught to students in the first semester of the computer music coursework.

Electronic and Computer Music II is an advanced course for students wishing to pursue independent projects in the studio. Each student must produce either a large-scale musical composition or a significant piece of computer music research. This course also focuses on developments outside of CMU, with required readings of important computer music papers and directed research into the writings of others, which may relate to and/or influence the students' own creative work. Both courses, Electronic and Computer Music I and II, are offered as part of the music curriculum; these courses are taught by a faculty team from Music and from Computer Science and are elected by students from all departments on campus.

2.1. The Adagio Language

The language used most frequently in the studio is Adagio. Adagio is based on the note-list concept of Music V, where each line of a score file describes one note or event. In Adagio, parameters of notes such as pitch and duration are described using self-identifying fields, so the order of parameter specification is arbitrary. For example, a half note G#4 followed by a half note A4 could be written:

```
h gs4
a
```

The order of the two fields in the first line does not matter: it could also have been written "gs4 h". In the second line, the duration is "inherited" from the line above and need not be rewritten. Similarly, the octave (4) is derived from that of the G# above, and can also be omitted from the program. In addition to the high-level symbolic notation used in this example, all parameters in Adagio can be specified in an optional low-level numeric format. For example, *Lff* means "the loudness is *fortissimo*", but this could also have been written *L127*, which means "use the number 127 as the peak amplitude".

We have found the Adagio language to be simple but versatile. The translator, combined with our synthesis hardware, can process a typical score and begin generating sound in seconds. A keyboard transcription program has been written to translate a real-time keyboard performance into an Adagio source file, which can then be edited and manipulated by the composer. In addition, several composing programs that generate Adagio as output have been written by students. The Adagio translator has also been adapted to generate tables for our real-time accompaniment programs, and another version of Adagio will soon provide MIDI output.

Adagio has some serious limitations, however. The primary one is that Adagio can only describe music in terms of a fixed set of parameters. While the set can be enlarged by modifying Adagio, we need a more flexible and extensible notation. The Arctic language, described below, is a step in the right direction. We are also developing a graphics-oriented score editor that will have the flexibility required to support serious composers and researchers.

3. The Solfege Lab

The Solfege Lab contains four Apple IIe workstations with Mountain Hardware music synthesizers, an Alphasyntauri keyboard, and an IBM PC. Music Department students use this lab in conjunction with their coursework to help develop skills in basic musicianship. The heart of the software in use here is Camus, an ear-training system developed at Carnegie-Mellon, which provides self-paced tutoring in pitch and rhythm perception.

4. The Computer Music Laboratory

The laboratory is equipped with a 68000/Multibus system for research in real-time control, and a prototype Musician's Workbench. The Musician's Workbench is currently under construction, and is described in more detail below. It is intended to serve as a vehicle for packaging the results of our research, and for making these results available to the campus community. Current

research is focused on (1) Arctic, a functional real-time control language, (2) Vivace, a rule-based AI system for composition using traditional harmony, (3) the sensor frame, a free-hand pointing device that allows users to point to and manipulate sonic or graphic objects, (4) real-time accompaniment systems, and (5) methods of acoustic analysis.

4.1. The Arctic Language

Arctic, described in detail elsewhere in these proceedings, is a language for the description and implementation of real-time systems. Arctic is especially adept at the manipulation of time-varying functions in a manner similar to GROOVE, but it also allows the programmer to build high-level descriptions of behaviors that can represent anything from the attack portion of a note to an entire composition. Arctic has very few biases toward particular ways of expressing music; it is primarily a powerful tool enabling the programmer to build up his own abstract concepts in terms of simple functions and operators.

Arctic is currently limited to the manipulation of piece-wise linear functions and we use Arctic to produce control functions for synthesis software and hardware rather than compute sampled audio directly. Direct computation of sampled audio is possible with Arctic, but it would be extremely slow.

We use a standard file format to represent collections of piece-wise linear functions. This format was originally used by an "Arctic calculator", a program that allows a composer to interactively manipulate functions and see the results displayed graphically. The results of this program can be fed to our 68000/Multibus system which drives one of our Apple II/Mountain Hardware synthesizers. (Incidentally, we use the same low-level synthesizer software and hardware here as in our studio, but the user interface is Arctic rather than Adagio or C.) This system provides rapid turnaround for experimentation and composition, but the audio quality is poor. We have improved the situation by writing a C program to simulate the functionality of the Mountain Hardware synthesizer while using large wavetables and floating point arithmetic. Thus, by standardizing on an appropriate representation for control functions, we have gained the ability to trade speed for sound quality and vice-versa.

We now have an interactive Arctic interpreter, that augments the calculator capabilities with function definitions and better graphics. This program uses the same external data representation, so it is also compatible with our software and hardware synthesis systems. We have benefited greatly by making many of our tools compatible with a single data representation for functions. Consider the following, which actually took place in our lab: a recently completed sound analysis program had been used to obtain a description of a trumpet tone. The description was output in our piece-wise linear format, which we then fed into our synthesis program to see if the output would faithfully reconstruct the trumpet tone. It did, so we decided to try the same functions on our hardware synthesizer. This time, the sound was quite different, and it was obvious that the synthesizer could not keep up in real time with the large number of ramps produced by the analysis software. Fortunately, we had a program to reduce the number of ramps (inflection points) by producing a piece-wise linear approximation. We used this program to process the analysis and fed the results back to our synthesizer. The synthesizer still failed to produce a realistic sound, and we suspected that the approximation software might not be working correctly. To test this suspicion, we used the Arctic interpreter to plot amplitude functions for the fundamentals, and they seemed close. To get a better idea of how close, we used Arctic to subtract one from the other. The resulting difference, that is, the error introduced by the approximation, was also plotted. The error was small, indicating that the approximation program was working correctly, so our next step was to resynthesize the tone, feeding the approximation functions to our software synthesis program. This test indicated that the approximation was good aurally as well as visually, so we returned our attention to the hardware synthesizer system to try to explain the problem. This example illustrates how tools can often be put to uses that were not intended by their designers. In a matter of minutes, we were able to use five independent programs to produce, manipulate, and examine data. We try to encourage this sort of creative use of software by making tools compatible with each other and by designing representations that are concise, flexible, and general.

4.2. Vivace: A Rule-Based AI System for Composition

Vivace models as closely as possible the human approach to writing music. In fact, the purpose of this research is to more clearly define the metarules governing the compositional process. Vivace's current task is to compose a four-part eighteenth-century chorale, using all the guidelines and constraints needed to ensure good voice leading, preferred doubling, effective choice of chord functions, convincing cadences, and judicious use of nonharmonic tones. Vivace is a modular system, designed so that the musical results can be studied with various aspects of knowledge implemented or ignored. The goal is to determine just how the skillful musician creates not just a correct chorale, but an aesthetically pleasing one. What rules take preference in conflict situations? How does the specific location within a musical phrase influence the local choice of notes? How much knowledge about good melody writing is needed in the inner voices? Answering these and other questions could significantly improve the teaching of such creative processes.

Vivace is based on the concept of a musical phrase. In contrast to previous attempts to harmonize a melody by selecting chords according to rules of probability, Vivace assumes that the role of any given chord is largely dependent upon its location within a musical phrase; that is, a IV chord at the beginning of a phrase serves a different function from the IV chord preceding a cadence. Chord selection, therefore, is not based simply on the value of the preceding chord.

If the musical phrase is adequately defined, and the way in which phrases are put together described, the computer should be able to generate a successful piece of music in any specified style. Although the eighteenth-century chorale is presently being used to test this hypothesis, other musical styles and other compositional conventions should be interchangeable without losing the effectiveness of the general phrase-constructing approach.

Vivace is a modular system containing the following components:

- Module #1: Melody Writer (optional)
- Module #2: Harmonic Rhythm Selector
- Module #3: Phrase Shaper
- Module #4: Chord Designator
- Module #5: Bass Writer
- Module #6: Voice Assigner
- Module #7: Nonharmonic Tone Adder

Presently, modules 2 through 7 are written in LISP and fully implemented on a DECsystem20 at Carnegie-Mellon University with musical results heard in the Computer Music Studio via translation into Adagio. The Melody Writer module is under development. Vivace is currently producing correct four-part harmonizations of melodies using triads in the home key, adding passing tones and neighbor tones in the tenor and alto voices to enhance the musical results. The system will accept melodies in any major or minor key using any standard time signature.

The implementation of a system to harmonize melodies correctly, using good voice leading, preferred doubling, and appropriate chords is in itself not particularly difficult, nor is it terribly interesting. But using this computer system to demonstrate the importance of various compositional practices and observing the results when specific rules are changed has become increasingly fruitful.

Vivace has been under development for a little over a year. We are just now beginning to reap the benefits of its knowledge. Although much remains to be done, Vivace has already produced a great deal of interesting information about the chorale writing process. We hope to eventually use this tool in the classroom to help demonstrate the impact of various procedures intuitively employed by the masters of the past. Bach has nothing to fear from our computer-generated music; he is still far ahead of us.

4.3. Real-Time Accompaniment

The task of a real-time accompaniment program is to listen to a live performance, to follow the performance in a score, and to produce an appropriate accompaniment according to the score. A successful accompaniment system must cope with changes in tempo, and it must ignore mistakes in the performance or errors in the

listening section of the system.

A successful system has been implemented and is described elsewhere in these proceedings. We are currently extending the system to handle polyphonic input from keyboards (the present system deals with strictly monophonic input, although it can generate polyphonic accompaniment). We are also looking forward to using our accompaniment software for the performance of a serious composition for computer and live performer. This will become possible when we complete the initial stages of our computer music system, which is described below.

4.4. Acoustic Analysis

We are presently investigating several techniques for pitch extraction and period-synchronous spectral analysis of acoustic instrument sounds. We hope to use the resulting software to study the dependence of spectra upon intuitive variables such as pitch and loudness. This may lead to better models for the synthesis of natural-sounding musical tones.

4.5. The Musician's Workbench Project

We have been frustrated by the difficulty of teaching, composing, and conducting computer music research on computer systems that were never intended for that purpose. We use machines that are optimized for text input and output, and which have no built-in sound synthesis capabilities. On the other hand, our experience with Adagio and Arctic tells us that our efforts in computer music could be substantially aided by a set of software tools that can communicate using a common representation for musical information. Furthermore, if we could combine these tools with a high-quality real-time synthesizer and a high-performance personal computer, we would have a system that would fill most of our needs and be available at low cost.

We are currently constructing the first version of a music workstation called the *Musician's Workbench*, which will combine:

- a 32-bit personal computer,
- a velocity-sensitive keyboard,
- the Bradford Musical Instrument Simulator developed by Peter Comerford and his colleagues (at the University of Bradford, West Yorkshire, U.K.),
- a *sensor frame* gesture sensing device to facilitate free-hand pointing and the manipulation of graphic objects.

The 32-bit personal computer is currently a Sun terminal, connected by a 10 Mhz Ethernet to a campus network which includes a centralized file server. It has a high-resolution bitmapped graphics display and a 60 megabyte winchester disk used for local cacheing of a working set of files. It runs under a modified Berkely 4.2 UNIX[1], using window management and file system software developed at CMU's Information Technology Center. We expect to move to machines with lower cost and higher performance when they become available.

The Bradford Musical Instrument Simulator is a 20 MFLOP table-lookup digital synthesizer that performs sample interpolation and in addition can interpolate between waveforms. This allows dynamic changes in spectrum at low cost. The device has hardware support for fast waveform generation, and multiple synthesizer boards can be attached to an inexpensive bus to allow many units to be controlled in parallel.

The principal software component will be a subroutine package for manipulating musical scores. Our current representation for scores bears a close resemblance to a semantic network; however, the representation is sufficiently restrictive to allow fast manipulation of scores. The representation is essentially a list of events, where each event is a list of attribute/value pairs. Additional structure allows score events to be organized into multiple hierarchies. As in Arctic, the basic representation scheme makes few assumptions about music, and it is up to programs that use the representation to establish the semantics of the representation. The primary use of the score manipulation routines will be to construct a flexible score editor. If our experience with Adagio and Arctic data structures is a good indicator, we expect a large set of tools to be developed for manipulating, viewing, creating, and performing scores.

---

[1] UNIX is a trademark of A. T. & T. Bell Labs

The *Musician's Workbench* will support computer-assisted music education, composition, and research, and will be fully integrated with the 5000-node personal computer network under construction at CMU. This means that students and researchers will be able to edit scores using any machine, most of which will be in offices and dormitory rooms outside our lab and studio. It also means that the same system used for music will serve other computing functions such as document production, programming development, and computer mail. This multiplicity of functions will greatly increase the utility of the overall system.

The cost of the workbench in the 1986 time-frame is estimated at about $6000 per workstation. It is intended to provide a system that students can use directly for studies in composition and orchestration. In addition, it will encourage the development of a variety of music instruction programs by providing convenient program access to a score editor and sound-generation hardware.

As a composition and performance system, the workbench will provide excellent tools and high-quality sound in real time. The Bradford Musical Instrument Simulator is already in use as a church organ, and as such it is quite impressive. We intend to develop software for orchestral and artificial instrument simulation to supplement the existing software for pipe organs, piano, and harpsichord simulation.

5. Conclusion

In less than two years, we have made considerable progress in establishing a computer music curriculum, constructing research facilities, and producing music. We owe a tremendous amount of thanks to the Computer Science Department for providing a superb environment for computer science research and for the opportunity to pursue our computer music goals. We owe equal thanks to the College of Fine Arts and to the Music Department for encouraging the development of our curriculum and for providing resources for teaching and performing computer music.