Fifteen Years of Computer-Assisted Composition

Stephen Travis Pope

Computer Music Journal, And Cnmat, Dept. Of Music, U. C. Berkeley.

P. O. Box 9496, Berkeley, California, 94709 Usa

Electronic Mail: Stp@cnmat.berkeley.edu

World-wide Web: Http://www.cnmat.berkeley.edu/~stp/

Abstract

This paper describes several generations of computer music systems and the music they have enabled. It will introduce the software tools used in some of my music compositions realized in the years 1979-94 at a variety of studios using various software and hardware systems and programming languages. These tools use a wide range of compositional methods, including (among others): high-level graphical notations, limited stochastic selection, Markov transition tables, forward-chaining expert systems, non-deterministic Petri networks, and hierarchical rule-based knowledge systems. The paper begins by defining several of the terms that are frequently used in the computer music literature with respect to computer-aided composition and realization, and introduces several of the categories of modern models of music composition. A series of in-depth examples are then drawn from my works of the last 15 years, giving descriptions of the models, the software tools, and demonstrating the resulting music.

Background

Composers of music in the Western tradition have always used the highest technology that was available to them (Pope, 1991a). Examples of this can be found throughout music history, be it J. S. Bach's use and advocacy of equal temperament (remember that Bach was a contemporary of Euler—the inventor of logarithms, on which the equal-tempered scale is based), or the heavy use of digital technology by composers and performers today. Because composition involves many technical processes and typical "engineering" trade-offs, there is no contradiction for a composer to use a computer and learn a software user interface or programming language; common-practice Western music notation is, after all, a highly technical and abstract description language for performance gestures and has very little to do with the resulting music or personal expression.

There are several good surveys of the use of computers in music composition (Loy and Abbott, 1985; Loy, 1989); the field is actively growing, as new informatics techniques find applications in music composition—recently artificial neural networks, the mathematics of non-linear dynamical systems, and genetic algorithms have been seen here. Some abstract representations—e.g., the use of instrument definitions and note lists—and classes of compositional algorithms—e.g., common procedural or stochastic models—have been in common use since the "dawn" of computer music in the 1950s and 1960s. This paper will survey several of these popular techniques (and a few rather obscure ones), and present concrete examples of the musical results and graphical user interfaces.

Algorithmic Composition and High-level Composition Tools

In the last 40 years, computers and informatics technology have found a wide range of applications in music composition, realization and real-time performance. Note, however, that the kinds of abstractions and tools used in these three areas—composition, production, and performance—are very different. Composers need software support for the refinement of incompletely specified ideas and the capture of abstract information early in the process of composition, and for the generation of scores or the transition from composition to production or performance (Pope, 1986, 1993b). The wide variety of processes involved in composition, and the vague and abstract nature of high-level musical material, both place special requirements on the design of software tools to support composers.

There is a wide-spread misunderstanding that all uses of software tools in the composition or production of music imply algorithmic composition. It is important to differentiate between a composer who uses the computer models of musical structures in the process of composition and organization of musical material, and the performer or producer who uses digital tools for the "later" tasks of music performance, production, or recording.

What I wish to focus on here is the use of computer-based software systems in the capture and elaboration of musical ideas—the use of computers as "composer's assistants" rather than as simple scribes or digital instruments. In this category, it is important that the computer have some software-based model of the musical structure or process. I will discuss below the different families of compositional algorithms and software-based composer's tools, and present examples of their use in my musical compositions from the period 1979-94.

Procedural, Stochastic, and Knowledge-based Systems

There are many ways to model complex data structures or dynamical systems using contemporary mathematical and informatics-based techniques. *Procedural* descriptions such as those expressed in common programming languages are widely used to describe the processes and policies of real-world systems in computer-based simulations. It is also possible to describe musical processes in a procedural manner, and there are numerous software systems that allow one to construct "musical automata" using straight-forward imperative procedural programming techniques. Music-oriented function libraries exist for most common programming languages such as C, Pascal, BASIC, Lisp, and Smalltalk. These generally allow the user to create some kind of musical data structure (often using MIDI parameters in lieu of a higher-level abstraction for musical parameters) and to store it to a file or play it in real time. There are also several kinds of music-specific languages that take other languages as their models, e.g., Barry Vercoe's *Csound* is based on assembly language; F. Richard Moore's *cmusic* on C, and William Schottstaedt's *clm* on Common Lisp.

Very complex dynamical systems cannot always be modelled directly in a procedural programming language, and modelers often resort to *stochastic* models that use grammars or other structuring paradigms together with limited random selection of values. There are many kinds of stochastic models, the most common of which are based on relatively simple structures (described in a procedural or static manner) that are elaborated with bounded stochasm (randomness). This can be as simple as declaring a musical structure (e.g., a melody) and adding some amount of random variation to its parameters to make it seem more "life-like." (Indeed, some commercial synthesizers and sequencers have a "humanize" function that does just this.) One commonly used higher-level stochastic model is that of generative grammars, which can be used to describe the "rules" of a given musical structure. Given a small set of rules (each of which describes a transition between a set of starting tokens and a set of final tokens), it is possible to generate musical structures that have a high degree of "internal consistency" (on some linguistic level at least). There is an active debate among music theorists as to whether or not music is “linguistic” in terms of the applicability of generative grammars to higher-level musical structures. Other common models involve bounded stochasm, where the computer is used to make a selection from a given set or range of values.

Several kinds of *knowledge-based* frameworks are also popular today, including rule-based, logic-based, and neural-network-based models. These systems differ from the previous types in that the “knowledge” of the musical domain is programmed in a declarative style—you tell it *what* there is to know about the task, rather than *how* to solve it. There is a debate (that I would like to avoid) in the literature as to whether neural nets indeed constitute a “knowledge-based” model, since there is no direct knowledge model involved, but rather a complex statistical model serves as their base. The literature of artificial intelligence (AI) and music is very rich, and numerous kinds of knowledge-based frameworks for music composition have been developed and used in real-world situations by well-known composers.

There are, of course, other basic modelling paradigms, such as statistical models—which may be classified as procedural or stochastic, depending on their structure. The above list is not intended to be complete, but rather to provide basic terms for categorizing the algorithmic composition approaches I will introduce below.

Description Languages, Automata, and Graphics Tools

The various stages of the composition process often call for several different kinds of tools or languages. Sometimes, a composer wants to use a formal language to describe a specific musical structure or process, e.g., for the purpose of annotation. In this case, he/she can use any of a large variety of formal languages, ranging from logic declarations to programming languages. There is a rich literature of “music input languages” that includes music description formats based on just about every known computer science formalism (Loy and Abbott, 1985; Loy, 1989; Pope, 1993a). Some of these languages are based on the declaration of simple, linear “note list” structures, while others allow the user to describe higher-level musical structures such as chords, performance gestures, and musical forms as well (Pope, 1986, 1989).

“Algorithmic composition” implies that the composer builds a program (the algorithm) that then composes (all or only some features of) the music. As described above, this description can take several forms, including procedural, stochastic, or knowledge-based software tools that are “generative” in the sense that they take some data as their input and generate musical data as output.

Graphical tools such as score editors based on common-practice Western music notation are widely used in music processing, but composers rarely have access to higher-level (i.e., higher-level than the “note”) notations. There has been some interesting research in the development of formal notations (i.e., those that are based on a strict encoding and mapping between graphical symbols and musical objects) for musical structures, and several composers have used these in the real musical settings (Loy, 1989). There is a continuum of options between complex score notations—pitch-time diagrams where the icons used have more information than traditional note heads and stems—and graphical programming languages for musical algorithms. This is a very interesting field of research in which there is much left to be discovered.

Examples

In this section, I will illustrate a variety of software-based composition techniques and composer’s tools that I have used in my own compositions. In the live presentation, I intend to show examples of the software user interfaces used for each of these pieces, and to play short musical examples that demonstrate the compositional structures or algorithms that are most heavily used.

I will not present the in-depth reference list or “picture show” in this paper, but instead refer the interested reader to the publications that are cited below. (I have published detailed articles on each of the tools I present here.) The examples are presented in chronological order, rather than by the level of sophistication of the technology they use.

Graphical Notations: *WAKE*

WAKE: Ten Tangents for Dance was realized at the SSSP (Structured Sound Synthesis Project) studio within the computer systems research group (CSRG) at the University of Toronto during1979/80. This piece originated as a work for organ solo, and had several generations of graphical scores including common-practice notation and a "piano-roll-style" score (i.e., a pitch vs. time plot) where color was used to denote organ voicing. This graphical score was "transcribed" for use by the SSSP's digital synthesizer rather easily. The further revisions to the score used several graphical notations that were specific to the SSSP system (Buxton et al. 1979), and for which tools were written in C. These notations went beyond the simple pitch/time diagrams and included structure trees, timbral notations, and other non-traditional graphical representations for which computer editors were built.

This is an example of realizing an "ad hoc" composition—where the compositional strategy was implemented without computer assistance—with a formal software-based tool. In this case, the score existed before-hand in a non-computer-oriented format, and was transferred to the computer by "mousing" it all in (as is commonly done today by composers who use MIDI sequencers). The computer tools played an important role in the "arrangement" of the original organ piece for a new "ensemble" of voice-like synthetic sounds.

Rule-based Forward-chaining Expert Systems: *Bat out of Hell*

Bat out of Hell: Stories for Dance was realized at the CMRS studio in Salzburg, Austria in 1983 using the "ARA" expert system in Lisp. ARA was developed to implement a specific model of the composition process in a tool that had machine-learning capabilities. It divided the process into the stages of generation of rhythmical patterns, selection of pitch sets and melodic phrase structures, the mapping of compositional properties to timbral control parameters, sound generation, and mixing/spatialization. The ARA tools allowed the composer to define a "language" in each of these domains where a set of adjectives was specified using musical examples, for example, "this rhythmic pattern is red, that other one is blue, and this one is very blue." As the user assigned adjectives and scales to musical material, ARA "learned" which attributes of the material were being influenced. It prompted the user in cases of ambiguity (which were plentiful).

After "teaching" ARA how to classify a set of materials and apply transformations based on user-defined adjectives, the user could then make requests such as "make the spatialization in this section rounder" or "use a smooth orange rhythm here." Due to the technology of the day (twelve years ago), ARA was a non-real-time system that generated Music-11 instrument definition and note-list files. These had to be compiled in "batch" mode. The piece *Bat out of Hell* can be viewed as a transition from order to chaos and back in which ARA started with a well-organized consistent set of material and was then directed to carry out a series of transitions into and out of less-organized areas. The timbre was kept very simple (a single-modulator FM bell sound) for both musical considerations and compute-time constraints.

Logic-based Petri Nets: *Requiem Aeternam Dona Eis*

Requiem Aeternam Dona Eis was realized at the CMRS and PCS GmbH in Munich during1984/85 using the "DoubleTalk" Smalltalk system. DoubleTalk used Petri nets (which are similar to state transition automata) where the states and transition rules were defined in a Prolog-like declarative logic format. Compositional processes were defined as Petri nets with a static structure (the layout of the net), pre-defined state capacities (how many of which kind of token can inhabit a given state at a time), and an initial "marking" (what tokens are where at time zero). There were a suite of interactive editors for "drawing" nets and "programming" their transitions, defining token types, and placing tokens in a net (marking the net). After this was done, one could "run" the net by letting a simulator step through the possible transitions like a rule base. Transition rules could have side-effects such as playing musical notes, or changing the rules of transitions or the marking of the net. DoubleTalk was also initially connected to a non-real-time software synthesis engine (cmusic), though later versions could generate MIDI output in real time. In *Requiem Aeternam Dona Eis* the DoubleTalk system was used to build musical structures that bear some similarity to the sounds of three sections of the text of the Latin requiem mass. These structures were then the subject of a set of variations that were generated by using the same net and initial marking but changing the transition rules.

Stochastic Procedural Systems: *Day*

Day: Installation was realized at Xerox PARC in 1987 using "EventGenerators" (Pope 1989) in Smalltalk. This is a real-time system that lets the composer/performer interact with a number of mixed procedural/stochastic "smart performers" or "hyper-instruments" that I call "event generators." Examples of simple event generators are chords, trills, and scales; one can, for example, define a chord given its root, type, and inversion, or a trill given its notes, duration and frequency, or a scale given its base, goal, and gamut. Stochastic event generators select at random from a given set of rata or data range, as in the example of a "cloud" that repeats the notes of a given chord according to a given rhythmic pattern. Higher-level procedural event generators use algorithms such as Markov tables or bell peal rules (as in the ringing of the changes) to generate their output. In *Day*, these event generators were configured ahead of time, and the performer could interact with them live. For example, there is one section that is defined as a set of fixed rhythms based on arpeggiated chords, but the actual chord notes are to be played by the performer in real-time.

Grammar-based Rule Systems: *Kombination XI*

Kombination XI is the first movement of *Celebration: Laments and Simple Truths for a Quiet Spiritual Place*. It was realized at the Vienna Music Academy, CCRMA, STEIM, and elsewhere from 1978-90 using "TR-Trees" and the "extended generative theory system" in Smalltalk (Pope 1991b, 1992). This system is loosely based on the seminal work by Fred Lerdahl and Ray Jackendoff on a general-purpose theory of any style of music based on generative linguistics. Their theory defines several kinds of hierarchies, from low-level event grouping up to high-level formal structures based on tension and relaxation (hence the name TR trees). For each of these hierarchies, they define rules for recognizing structures and defining the well-formedness of alternative structures. The TR-Trees system is essentially a set of knowledge bases that implement a specific music theory (my style, as it were), and allows one to use this knowledge in generating (rather than analyzing) music. The expert system engine was the same for all knowledge bases. In *Kombination XI* these rules govern the processing of spoken text as musical material so that a tight symbiosis is achieved between the text and musical structures. Both the musical and text-like sounds in the piece are derived from the sounds of the spoken voice.

Hybrid Multi-tool Environments: *Paragraph 31*

Paragraph 31: All Gates Are Open was realized at the Swedish Institute for Computer Science (SICS), and the EMS Studio in Stockholm during 1992/93 using a wide variety of tools written in Smalltalk and C. My current feeling is that a composer needs to be able to use many different abstractions, notations, and algorithmic tools within any one composition, so I have built a framework that includes all of the components mentioned above, and more. The current hybrid system—implemented within the Musical Object Development Environment, or MODE—includes several kinds of extended graphical notations, a rich set of procedural, stochastic, and algorithmic event generators, the TR-Trees knowledge bases, and a DoubleTalk-like network-based tool. The system can also import data from and export data to a variety of external programs such as MIDI sequencers and real-time digital sound file mixers. This system was used along with various third-party C-language tools in the composition and realization of *All Gates Are Open,* in which the basic structures were described using TR-Trees that generated the specific gestures represented in terms of event generators. These generators were edited using graphical notation tools. The Petri net subsystem was used for the mapping of the actual performance rules that governed the system's "interpretation" of the raw musical data. The final production was described using several Smalltalk- and C-based tools that generated and edited mixing scripts for a real-time digital signal processors at EMS.

Conclusions

My current development effort is aimed at extending the MODE to include still more tool paradigms for new notations, and alternate procedural, stochastic, and knowledge-based algorithmic composition abstractions. My works-in-progress (of which there are several) involve new tools based on context-free generative grammars, other kinds of rule bases, and performance mapping using fuzzy logic.

I strongly believe that, in the future, broad multi-paradigm composition environments will replace the current crop of single-purpose, single-representation tools, and will provide musicians with ever more flexible, scalable, and multi-level facilities for music composition, realization, performance, and production.

References

Buxton, W. et al.,1979. "The Evolution of the SSSP Score Editing Tools." *Computer Music Journal* 3(4): 14-25. Reprinted in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press.

Loy, D. G., and Abbott, C. 1985. "Programming Languages for Computer Music Synthesis, Performance, and Composition." *ACM Computing Surveys* 17(2): 235-266.

Loy, D. G. 1989. "Composing with Computers—A Survey of Some Compositional Algorithms and Music Programming Languages." in M. V. Mathews and J. R. Pierce, eds. 1989. *Current Directions in Computer Music Research*. MIT Press. pp. 291-396.

Pope, S. T. 1986. "Music Notation and the Representation of Musical Structure and Knowledge." *Perspectives of New Music* 24(2):156-189.

Pope, S. T. 1989. "Modeling Musical Structures as EventGenerators." *Proceedings of the 1989 International Computer Music Conference*.

Pope, S. T. 1991a. "Music Representation, Compositional Methodologies, and Computers." *Proceedings of the Second Symposium on Music and the Sciences*, Seattle: University of Washington.

Pope, S. T. 1991b. "A Tool for Manipulating Expressive and Structural Hierarchies in Music (or: T-R Trees in the MODE: A Tree Editor Based Loosely on Fred's Theory)." *Proceedings of the 1991 International Computer Music Conference*.

Pope, S. T. 1992. "Producing *Kombination XI*: Using Modern Hardware and Software Systems for Composition." *Leonardo Music Journal* 2(1): 23-28.

Pope, S. T. 1993a. "Music Composition and Scoring by Computer." (Invited chapter) in G. Haus, ed. *Computer Music and Digital Audio Series, Volume 9: Music Processing*. A-R Editions. pp. 25-72.

Pope, S. T. 1993b. "Real-Time Performance via User Interfaces to Musical Structures." *INTERFACE* 22(3): 195-212.