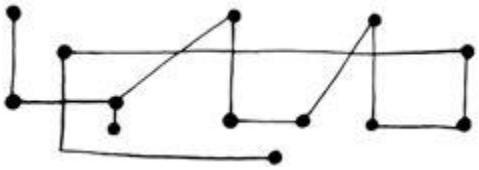
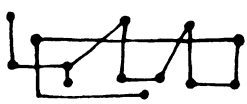


MUSIC NOTATIONS AND THE REPRESENTATION OF MUSICAL STRUCTURE AND KNOWLEDGE



STEPHEN TRAVIS POPE

INTRODUCTION

THIS ESSAY POSES several questions about new music notations and presents several examples thereof. The background sources for these investigations were discussions over the space of several years at CMRS and in Munich with Irmfried Radauer, Hans Strasburger, Clarence Barlow and the late Stephan Kaske (among others) and the realization by the author of a series of ballet pieces between 1979 and 1985 with varying degrees of computer involvement in the processes of composition and notation.

The presentation of these questions is as a series of examples and evaluations of new notations for structuring processes of time for musical performance by computer. The realizations discussed here have used computer software (programs) for music analysis and graphics-assisted editing of scores that has been

developed and combined with computer (software or hardware) sound synthesis in several different installations.

The motivation of these considerations is to help in the specification of a new musical software toolkit called *DoubleTalk* that should eventually support several of the basic notational paradigms demonstrated here in a flexible and integrated programming environment for music notation design and input. Two other papers by the same author (Pope 1986a and 1986b) discuss the related aspects of knowledge representation for process description languages and a survey of several music input languages for the pre-specification phase of the *DoubleTalk* system.

As a manner of introductory information, one must refer to the continuing discussions printed in the *Computer Music Journal* on the relations between recent advances in artificial intelligence (AI), digital signal processing (DSP) and music theory. Much interesting work has been reported here (and in the proceedings of the International Computer Music Conferences) on new music input languages (MILs) for using computers in music notation and synthesis.

In addition to this, a recent (June, 1985) special computer music issue of the *Computing Surveys* of the ACM (Association for Computing Machinery) included important articles from several well-known authorities in the field (see especially Roads 1985 and Pennycook 1985).

Notations of Music

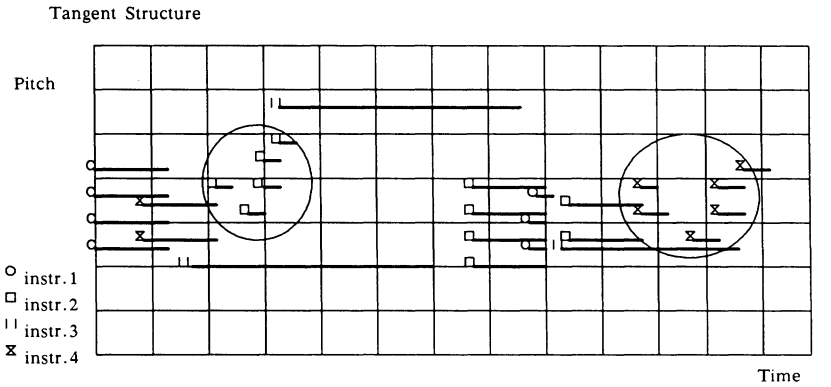
The history of the notation of occidental music has been deeply linked to the need to transfer musical ideas from a composer to a performer. This is largely unknown in many other musical cultures, where these two roles are traditionally filled by one-and-the-same person. The advent of sound recording and especially of electronic music techniques during this century have in many cases changed the relevance of music notation (see the numerous 'read-along' scores for electronically-realized compositions). This process is being further accelerated by the increasing appearance of computer-based composition systems or composer's assistants.

The recent literature on the subject of compositional models of music in the form of computer programs leads one naturally to question the new meanings that music notation can take on. A short list of the notational paradigms for musical material could be:

notation of:	examples:	
composition parameters	(score (?))	(sketches, block diagrams, . . .)
performance parameters	(parts)	(tabulature . . .)

first-level analysis of the composition when the score is projected during a (dance) performance.

(first movement (ew) model & cell type—simplified)



EXAMPLE 1: FROM *Wake* (VIENNA, TORONTO: 1979–80)

The score pages of the ten movements of *Wake* are simple pitch versus time graphs (timelines as in CMN) where grouping is used (see the circles in the figure) to make altered repetitions and variations more obvious and the "notes" are complex structures called cells. The structure of a single cell is shown in Example 2 along with its icon or graphical object notation.

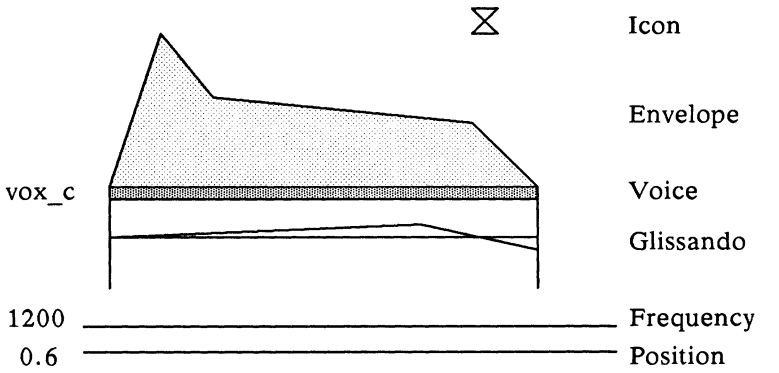
It is important to note that a fair amount of hand expansion of this notation was necessary in the realization of this composition, that is to say the computer programs could not parse this idealized notation (in 1979) and another level of notation (a compromise) was necessary. In the case of *Wake*, this just meant that a very large number of partially redundant instruments (identical except in one parameter) were needed to carry some of the additional information that was originally encoded as properties of fewer objects with more complex notes in the original (unplayable) score for organ.

Because of the complexity of the cells, the mappings of this idealized notation onto the two different versions of the real executable score (one for organ and one for computer) involved expanding these cells into many more instrument types than were actually involved. This shows some of the limitations of CMN-like notations for extreme or exotic instruments or for notating structures or larger forms. The circled-in areas can be considered as macros or motives used in one or more movements of *Wake*.

Wake is a mixed real-time piece in that the studio where the computer version was realized (SSSP at the University of Toronto) allowed a composer to control some of the music parameters (tempi, dynamics, etc.) at performance time by

the digital synthesizer using a *conducting box* while the computer played the notes of the score. This means that varying interpretations of the same computer score would be possible and that the composer could determine in advance the range of real-time control available to a conductor/performer at any particular moment.

The objects (instrument models) are cells that have parameters and functions as in traditional computer music compilers. The ObjEd (object editor) software at SSSP allowed defining them using a similar notation to that of Example 2. The voice selection within a cell determines the synthesis method (f.m., fixed waveform, etc.) and waveform functions for the basic timbre elements of the instruments.



EXAMPLE 2: *Wake* CELL STRUCTURE

The realization of *Wake* at SSSP involved using their very comfortable, graphics-based user interface (see Buxton et al. 1979, among others) to enter the ten movements from the organ score and developing simple instruments for the SSSP synthesizer using voice-like timbres. The system could generate an exact ASCII (textual, that is) score from these graphical event lists and object descriptions as in more traditional computer music languages.

The SSSP system has served as model for a plethora of newer score editors (see Yavelow 1985 for examples) that use strict timeline notations, either with CMN or proportional (piano-roll) notation. The separation and differentiation between object (instrument or patch) and event (notelist) editors seems fairly ingrained to most of these systems.

Regardless of the possible notations of higher structural elements, timelines and object/event list notations are obviously very useful for generating performance scores and/or music compiler notelists. Much of the discussion and comparison of different process notations centers around their behavior when

compositional knowledge	(analysis)	(algorithms . . .)
perceptual parameters	(recording)	(verbal description . . .)

(many other possibilities . . .)

The important questions or criteria in a simple taxonomy would be:

For whom are we notating?
Is the goal transparency of compositional structure/macrostructure/methods?
or,
Is instrumental/perceptual relevance of the notation more important?
To what end does one want to notate?
Does one notate performance parameters (i.e., technical information as in Common Music Notation (CMN))?
or,
structures and structural notation (i.e., motivic variation or grouping)?
or,
information relevant to an analysis (i.e., tonalities or interrelationships)?

Another question is whether the possibility of exact regeneration is a criterion for performance-oriented notations (the old question of completeness of notation, i.e., is there an inherent qualitative difference between very "low-level" versus very "high-level" notations?).

Designing Musical Notations—Models of Music

Based on these introductory questions one can observe the collection of problems raised when one decides to depart from the world of CMN and try to fit the notation of a particular piece to its needs, structure, techniques or compositional (developmental) history.

One of the most interesting new possibilities afforded by the use of computers as musical instruments (though still too seldom used in computer music) is the ability for a composer to design a new notation for a specific composition and then write a computer program to "read" (programmers say parse) that notation and generate sounds. The exploitation of a composer's fascination with new notations is then bounded only by his ability to model (program) a parser for each one.

The main portion of this essay presents the main properties of several models of time-domain processes, together with examples of each of them from specific compositions. Further examples can be found in the new music literature where composers have increasingly distributed works in the form of algorithms and

input data used to generate the final composition (as in much aleatoric music) or as multiple combined representations (modules and module selection criteria, for example).

TIMELINES—CMN AND PIANO-ROLL NOTATIONS

The most familiar notation is certainly Common Music Notation (CMN), which falls into the class often called timelines or object/event-list models. Some properties of this class are listed here.

Properties:

- The note-level organization is visible (as a time-ordered sequence of events).
- There is a strict differentiation between objects (instruments) and events (notes).
- There are few available higher-level syntactic and semantic structures (perhaps repeats or key signatures?).

Problems:

- Any recognition of higher musical forms demands a good understanding of the notation and probably the music theory behind it.
- There are problems with running out of icons or symbols to denote special features of objects. Text additions to the score (i.e., *legato*) are often used.
- A complex score has many parallel timelines (which is hard to read).
- The strict object/event (instrument/note) time-ordered separation makes meta-level features (macros) and phrase attributes difficult to impossible to notate.

The musical literature is comprised almost completely of timeline notations of compositions. The pros and cons of these methods are well known and have been exhaustively discussed elsewhere in the literature. Modern software techniques for managing event lists and object descriptions are found in the well-known systems described in Buxton 1978, Decker and Kendall 1985 or Abbott 1981.

Example 1 shows a simple extension of the traditional CMN-type timeline used in the ballet *Wake* (1979–80) whereby each element (note event) carries several extra pieces of information along with it (i.e., glissando function, position, etc.) without the need to complicate the score with text information. Simple event grouping is used to show motives and their variations as well as to present a

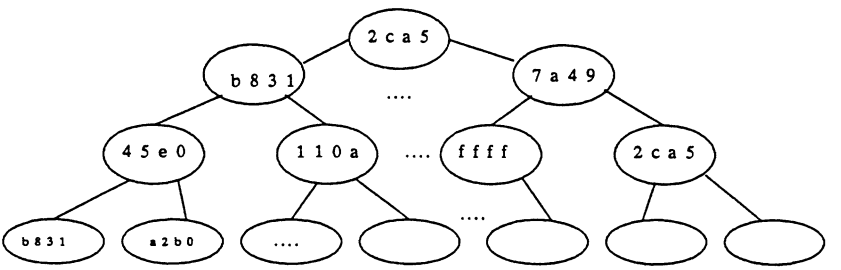
mapped onto event lists. The possibility of defining unique, repeatable and/or reversible mappings for generating notelists with process notations is also a determining factor in the specification of any new graphical notation, with computer tools or other means.

SEQUENCES AND TREES

The simplest way of structuring timelines is using sequences, series or rows. The management of these elements in a hierarchical manner in trees seems to be worth examining. The methods of using inheritance to describe timeslices of a musical structure by giving them paths to different lists that determine local (note level) attributes by specializing on more global (phrase, voice or section) score features is seen in sequencers and selection-based notations (see Krasner 1980 or Truax 1977).

The second score example here (Example 3) is of the stream notation used in the ballet piece *"4"* (1980–82). Here there is a tree structure of different basic motives or sequences that are mapped onto the four voices. This notation was executed by hand down to an intermediate level and then expanded by a computer program for playing in real-time on an early digital synthesizer and conducting box built by Didier Roncin at IRCAM in Paris in 1980. The conducting box allowed programmed selection of several parameters (octave, position, tempo, motive, . . .) by changing node links in the tree during a live performance on the synthesizer.

(second movement sequences—one voice, one parameter)



EXAMPLE 3: FROM *"4"* (PARIS, SALZBURG: 1980–82)

This notation was designed to show the simple (almost-minimal) sequence structure using hexadecimal (base 16) numbers to denote the selection parameters for the four features of each sequence (motive, voice, octave and either tempo or dynamic). This type of notation is a mixture of timeline and state machine and could be used for any strict sequence composition method.

Additional complications would arise however, if one wished to have full flexibility in notating variations between executions of the sequences or between mappings of the parameters to specific instruments. In the case of *"4"* however, these aspects were part of the performance and hence were not included in the score. This is an example of an idealized score that notates only the pure structure of the composition and where any live performance can stray significantly from the compiled score in the computer's memory or a printed version.

Trees used to denote sequences and motivic or serial composition can be very useful but offer few of the concrete mappings onto performance parameters seen with timelines. This could be summarized so:

Properties:

Hierarchical notation of parameters and sequences, series, rows, etc. is allowed.

Notation of control and/or hierarchy-determined parameters is possible.

Simple (one-to-one) mapping (expansion) onto timeline notation is often possible.

Problems:

There is often no concrete relevance of sequence or motive objects.

Notation of variation or derivation of related material is difficult (thick forests of trees are difficult to manage).

There are many possible mappings of abstract trees onto timelines or other notations.

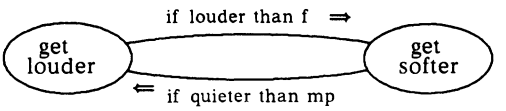
This notation of sequences can be viewed as a mix of timeline and structural notations. While one can attach a strict temporal ordering to the left-to-right sequence of the lowest level of the tree, it is not necessarily the case that all trees are performed in this manner.

The use of tree notation for parameters other than pitch and time is another possible departure from timeline notation. One can even envision the use of several parallel-running trees for various parameters of a composition or for separate voices (as in the case in *"4"*).

One of the main advantages of the use of sequences is their adaptability to modern computer hardware and software realizations. Modern digital sequencers with some measures of computer control often allow some form of hierarchical sequence editing that can likened to this tree notation. The comments that follow about mappings between similar notations apply to trees as well. One could design systems of many parallel trees or subsequences for complex control and/or methods of deriving families of trees (entire pieces) in hierarchical structures.

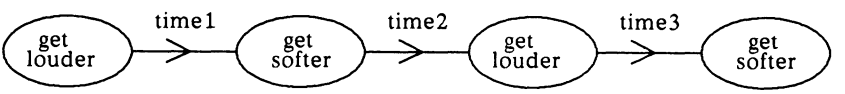
STATE MACHINES AND TRANSITION DIAGRAMS

Another well-known method for designing or representing complex processes is state machines, also called state automata or transition diagrams. A small, two-state machine that represents a crescendo-decrescendo between *mp* and *f* could be drawn like this:



This diagram would mean that we have two *states* (getting louder and getting softer) and two *transitions* (when louder than *f* and when softer than *mp*) for this simple system. The important concepts are those of states and their variables (loudness), of external controls and of state conditions that cause transitions between states.

From this example we can see that one can often mix time-implicit and time-free machines. We can extrapolate from the above net that this machine should toggle periodically, giving us a more-or-less regular switching behavior that could also be notated with a linear state machine. The process of expanding this machine into a linear (timeline-like) machine would only entail determining when the actual transitions occur and making new states for the sequential occurrences of the two possible states described above. The resulting linear machine would resemble:



One sees here that we have gained a clear time representation at the cost of the clear control and transition condition representation afforded by the first machine. Linear machines are very trivial to map onto object/event timelines so a first level of mapping and abstraction is available. It is also possible to describe relationships and transformations between different state machines. In this way one could imagine methods of making successively abstract machine models of an existing musical structure or designing one by stepwise refinement and mappings of the machine models.

The other modeling methods described below can often be represented as state machines on some level of expansion or simulation. This makes them useful in many cases as middle-level notations that can be generated by more abstract models and with which one can generate either heirarchical structures or event lists. The properties of state machines in general could be summarized so:

Properties:

The model has well-defined inputs and outputs, control paths and actions.

There is no certain musical relevance of controls (time sequencing?) (This can be seen as an advantage).

Many parallel state diagrams are possible (but is this legible)?

Problems:

Time dependencies are not inherent in a control diagram (expansion).

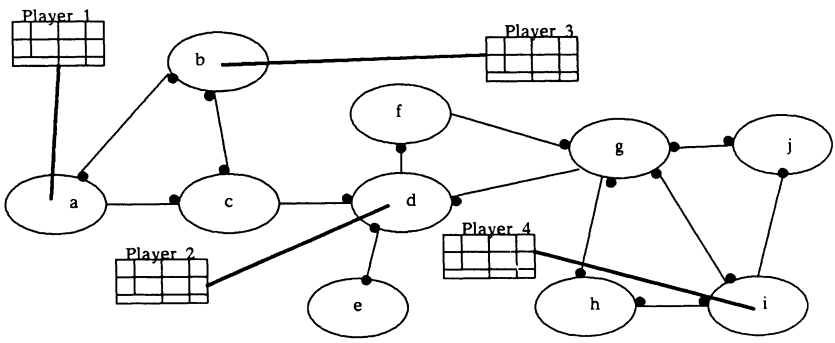
Complex, fully marked machines are quite unreadable (other than by machine...).

Transition diagrams are often used in many areas of engineering to aid in the visualization of the functioning of many types of complex structures, from digital hardware to the management of large organizations. There have also been several applications of these methods to music notation in the past.

Example 4 shows an excerpt from an abstract machine that was used to generate the machine score of the ballet *bat out of hell* (1983). The readability of this notation was of secondary importance in the creation of the piece. The notation denotes the method of selection of compositional parameters for the structure of each of the movements.

The specific states of this machine each map onto a set of parameters for the instruments of the four semi-independent players. The original version described four separate machines, one for each voice or process. These were then combined in the final production (expansion) score.

(second part machines)



EXAMPLE 4: FROM *bat out of hell* (SALZBURG: 1983)

Each state in this diagram represents a set of probabilities and distributions for a process (i.e., player) that inhabits that state. The players are simple note-generating processes whose output is influenced by their present state and their historical path through the machine in the current movement. Transitions between states are determined by conditions within the player modules and can also entail altering their internal status in the process of moving to a new state.

A series of simple LISP programs (the ARA system) were written for *bat out of hell* that allowed one to design a machine like the one shown and then enter it into the software as a series of short declarations and state variable values. The basic model used to execute a machine and generate a notelist was the heart of a LISP resolver program for theorem-proving that used the machine description and state loading as starting premises. This LISP program then generated as its output notelists for a music compiler (in this case musicll).

As above, part of the realization of the score was expansion by hand into a more machine-readable form. The fully marked (notated) machine for the above example is approximately one hundred lines of fairly dense LISP input programming. The same would have been necessary (and is common) if a score for performing musicians or other event list format were to be produced.

While state machines are very useful in notation of complex processes and models thereof, the limitations that apply to timeline and hierarchical notations are found here as well, namely the difficulty of notation of control and structurally-relevant parameters and the existence of many mappings when expanding machines onto timelines. Nevertheless, the flexibility of this structuring method makes it attractive for models that could subsequently (and preferably automatically) be mapped onto other representations.

CONTROL-FLOW MODELS—YOURDON ANALYSIS AND INFORMATION CONTROL NETWORKS

More radical ways of modelling time processes were developed by Edward Yourdon (see Yourdon 1981) and involve modifications and extensions of the state machine model. Here one allows notation of transitions based on the state of the process (i.e., loudness in the above state machine example), on arbitrary inputs to the system (triggers from outside for example), or on status information that is not directly part of any particular state or input. Several different basic models arise out of this work, including control-flow tables and machines, control-marked automata, decision tables and various other modelling and simulation methods.

The class of control-flow mappings that can be constructed includes state machines and sequences as well as mixed timeline/parameter and selection range notation systems. This allows one to easily define arbitrarily complex cross-relations between states and/or inputs of several parallel-running automata in a consistent and extensible manner. Related theories that can be or have been applied

to the design of process description languages (of which music notations are special cases) are RTS (structured analysis of real-time systems, Hatley 1984), ICN (Information Control Networks Ellis 1979) and KI-Nets (*Netze aus Kanten und Instanzen* (nets of edges and instances) Craemer 1978).

It will perhaps suffice in this description to compare some of these possibilities to the previous models and present some examples. They offer at any rate a flexible and interesting way of thinking about time-ordered functional relations and meta-level descriptions thereof, especially for music notation and analysis.

Properties:

There are many possible mixtures of object/event and state diagram models.

They are inherently multi-process in nature and can notate arbitrary cross-relations between controls, internal signals and outputs.

Flexible assignments of levels/outputs/representations to a model or diagram level or parameter are possible.

Problems:

There are (as with state machines) many possible different mappings of a particular result (i.e., simulation model output) onto CMN or other event-list notation.

There are also many mappings of control-flow and related compositional methods onto more common ones (e.g., deterministic, serial, Markhov, granular, . . .).

There are other possible models of related notations to be found in the literature and especially in recent computer music and artificial intelligence research results.

Example 5 shows an excerpt from a newer piece, *Requiem aeternam dona eis* (1985), where a mix of state machines and control-flow description was used to define voices tracing paths through several multi-dimensional aspect spaces (i.e., parameter fields) in a specified time interval. The figure shows two voices in a space describing two aspects for one movement whereby each line segment is of equal duration (six seconds) and the small filled circles represent fermate or pauses. The functions along the axes of the graphs are selection range width functions. The full score consists of several (in this case four) overlaid aspect-path graphs for each of six movements, where the constraints on the paths are notated by the oval-shaped areas in the diagram and the path selection was performed with the aid of LISP program.

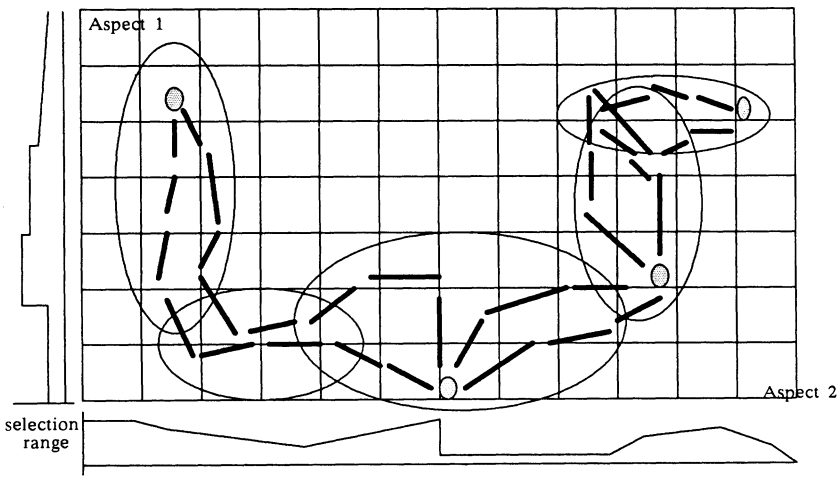
This method bears a distinct similarity to that of the POD and PODX systems (see Truax 1977 and 1985) when one relates aspect 2 directly to time and ignores

segment and path lengths to come up with a proportional selection range notation. There are certainly other interesting possibilities in this area.

The software front-end for this notation used LISP programming techniques with property lists and frame slots to define aspect spaces and transitions and simple database fetch operations with active property descriptions to write the scores. The input data was entered by coordinates and range spaces with a simple function editor.

In this case as well, some of the score expansion was done by hand (down to the level of constraint areas) and a machine-readable score for a music compiler was generated by software from there. This is the last composition by the author (he hopes) whereby the score was generated using mixed computer and manual processing (expansion).

(third movement, first phrase, two paths)



EXAMPLE 5: FROM *Requiem aeternam dona eis* (MUNICH, SALZBURG: 1985)

Control flow analysis offers a family of process description and notation possibilities, this example being just one. Other related models include control and object flow tables, parameter determination maps and input/output analysis models. There are certainly many useful and interesting examples still to be explored in this area. There are also concrete relationships between some of these models and the following network possibilities, but this is outside of the scope of this essay.

PETRI NETS

Researchers in computer music have also used the Petri network theory (see Petri 1976 or Reisig 1982) to aid in music analysis and/or notation. The basic principle of this theory is that the primitive functions of process flow are denoted using graphical means as in the transition diagrams above.

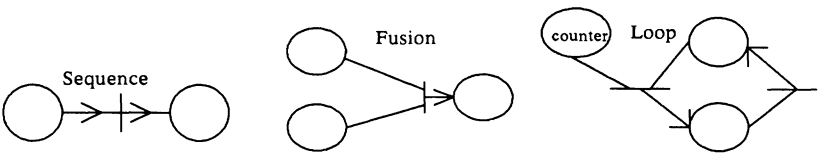
The distinction between these nets and state machines is that Petri nets denote states as objects that can be sequenced or selected based on their internal behavior (or variables) and transitions as separate entities whose "firing" (transition, that is) can be described based on the condition of some variable of a state or on arbitrary control factors.

This definition has the interesting side-effect of the incorporation of control and decision-related information into the "objects" that flow within a Petri net. Another of the properties of the nets useful for representing music is the possibility as seen above of having several different net descriptions of a single musical process that can be mapped onto each other using strictly-defined two-way mappings.

This allows one to use Petri nets for displaying a score with varying levels of specificity using the same notational methods and primitives. The Petri net score section in Example 7 shows a segment of a Bach canon at motive-level resolution and the entire canon at a very high level of abstraction. The mapping between related nets is a very basic part of working with these modeling and notation methods.

The musical applications of Petri nets require several new low-level primitives for notating time processes. The most basic net modules for process description are sequence, alternative branching (transition-determined decision), conjunction and forking. More complex process notation requires ways of specifying linkers and synchronizers as well as the higher-level methods of looping and signals to implement *wait* and similar new primitives.

As an example of the musical applications of Petri nets, one should include several figures taken from the discussion in "Music and Causality" by Degli Antoni and Haus (1982). Several of their primitive object types and functions are illustrated in Examples 6 and 7.

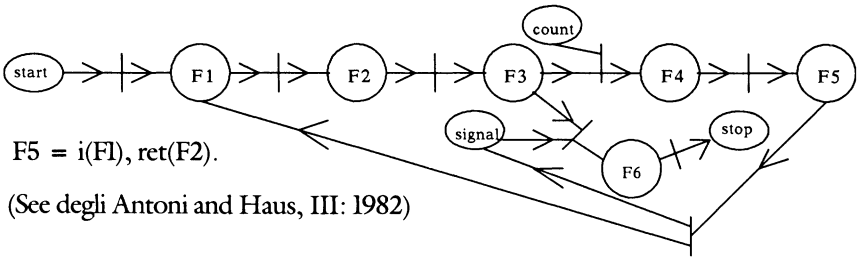


Printed in the *Proceedings of the 1982 International Computer Music Conference* and in Herbert Bruhn, Rolf Oerter, and Helmut Roessing, *Musikpsychologie: Ein Handbuch in Schlüsselbegriffen* (Munich: Verlag Urban und Schwarzenberg, 1985). Used by permission of the publisher.

EXAMPLE 6: EXTENDED PETRI NET ELEMENTS

Extending the primitives of the general net theory, one is able to map simple musical structures at several levels (notes (events), motives or phrases (minor timelines), or sections or thematic segments (complex event lists or sections of linked timelines), for example) onto nets that can then be used for analysis or generative modeling through simulation. Example 7 shows an excerpt of the flute voice of the opening of the Perpetual Canon from the *Musical Offering* of J.S. Bach, copied from Degli Antoni and Haus 1982.

The nodes labled F* are notelists of the relevant motives each several measures in length. Separate, linked linear sub-nets describe the exact notelists for these nodes, and operations within a net (i.e., variation or motive derivation) are also available. Node F5 for example, is notated (F5 = i(F1), ret(F2)), as a combination of some function of F1 followed by a transition to a return of F2.



Printed in the *Proceedings of the 1982 International Computer Music Conference* and in Herbert Bruhn, Rolf Oerter, and Helmut Roessing, *Musikpsychologie: Ein Handbuch in Schlüsselbegriffen* (Munich: Verlag Urban und Schwarzenberg, 1985). Used by permission of the publisher.

EXAMPLE 7: BACH, *Musikalisches Opfer*—SINGLE VOICE AT MOTIVE-LEVEL MODELLING RESOLUTION

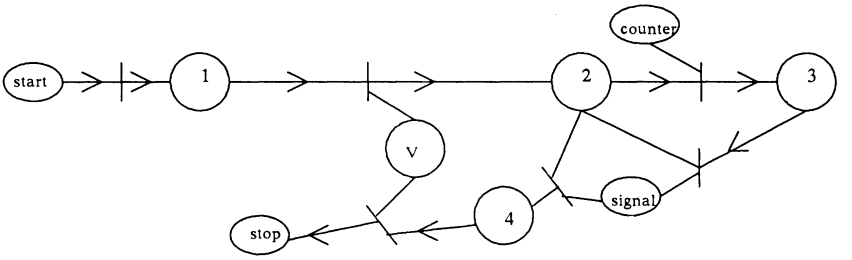
To generate the correct flute score, we set the counters and signals and load the lower nets with their event lists. The simulation should be able at this level to generate a transition firing sequence (note the terminology from expert systems technology) that resembles:

start Fl F2 F3 F4 i(Fl) ret(F2) Fl F2 F3 F6 stop.

This technique of generating notelists from net simulations is not yet so attractive relative to the above-mentioned modeling strategies (timelines or trees) until one observes several of the side effects of using nets. The easy mapping between nets and its guaranteed reversibility, the extensibility of the Petri net notation and the various simulation strategies available will be discussed below.

Example 8 (also taken from Degli Antoni and Haus 1982) is a very high-level net for the entire canon where one sees the coarsest level of modeling. This net

would serve as the root of a hierarchical structure of subnets that include the foregoing examples and would be developed until it extended down to note level in all areas and could be simulated in full.



Printed in the *Proceedings of the 1982 International Computer Music Conference* and in Herbert Bruhn, Rolf Oerter, and Helmut Roessing, *Musikpsychologie: Ein Handbuch in Schlüsselbegriffen* (Munich: Verlag Urban und Schwarzenberg, 1985). Used by permission of the publisher.

EXAMPLE 8: BACH, *Musikalisches Opfer*—CANON AT HIGHER LEVEL OF ABSTRACTION

Petri nets can also be viewed in a trivial case as state machines and parallel related nets can form arbitrary hierarchical structures like trees or control-flow diagrams. Petri nets indeed can be used to model almost any process description metaphor, language or paradigm.

PREDICATE TRANSITION NETS

Several groups have been working in recent years on extending Petri's general net theory to be more applicable to areas where flexible process modeling and/or simulation are desired. Most of these developments involve attaching additional information to the net's components, either with inscribed object types that flow through the net, with object capacities for the nodes (states) and edges (vectors or paths) or with attached logical or procedural information (programming) for the transitions' conditions and actions.

One of the most interesting results of the work in this area are Predicate Transition (PrT) nets. These are a simple combination of Petri nets with predicate logic programming used to describe transitions and with complex objects used for specifying states, edges and their capacities.

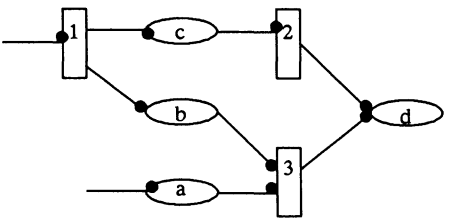
In a PrT net, a state (called a predicate here) includes information (logic- and object-oriented programming) about the types and numbers of objects it can hold, and a transition contains a condition part that determines when the transition will occur and an action part that can be used to alter objects or to pass them on to subsequent predicates.

Anyone familiar with rule-based expert systems technology will see that a transition resembles a rule, whereas predicates have their roots (their relatives) in logic programming. What is possibly more interesting is the extension of the more simple models (i.e., state machines) in the sense that we speak of some abstract objects that flow within this net, reside in predicates whose conditions (stated in terms of these objects) can be fulfilled and that can be used in the condition parts of transitions and/or processed by their action parts upon firing.

The full description of a PrT net includes its elements and connections as well as the object types and capacities of its predicates and the programming of its transitions. This means that we have a mixture of graphical and textual information used to describe a model but at the same time that very complex models can be built with just a few elements (together with their programming).

In Example 9, one could interpret the marking of predicate "a" as a pseudo-predicate-logic (prolog-like) definition that describes a note list or collection of events as being a loud C-Major chord in some inversion. When active, this predicate can hold several objects of this type (or the other predetermined object types mentioned) as described in the capacities.

The transition labeled "3" has as its condition the statements that the time (t) is greater than twenty-one units (seconds, for example), that there are currently at least four voices in the current chord and that it is major. When this transition switches (fires), it transposes the active chord by some value (4) and plays the chord with a pause following it, that is it writes some events on the output stream of the net simulation. The marking of the edge between "a" and "3" would determine how many of what type of objects are passed between them by the firing of "3."




```
predicate a = maj(chord), inv(chord),
              tonic(chord, C), loud(chord).

capacities a = num_voic(6), tim(24),
               pauses(6), repeat(2).

3 = IF (t > 21), (maj(chord)),
       (num_voic > 4).
    THEN (transp(chord, 4)),
         (play (chord, tim)),
         (play (pause, 2)).
```

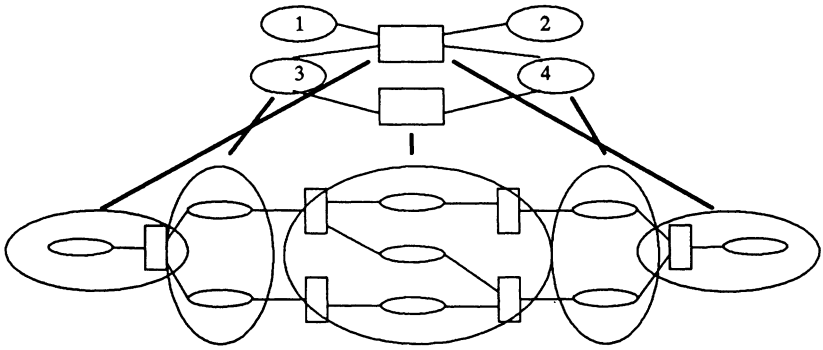
EXAMPLE 9: PrT NET SHOWING A SIMPLE NET SECTION AND THE MARKING OF A PREDICATE AND A TRANSITION

PrT nets can be used to describe all of the models discussed above and offer a very powerful and simple basic notation for systems where one wishes to be able to mix among these various possibilities. More complex nets become, of course,

fairly unreadable but there exist excellent means of managing them and incrementally developing a marked net on several levels with interactive simulation possibilities.

In the area of information control modeling and operations research one often sees PrT nets applied to the modeling of a set of processes by starting from descriptions of the system as seen by each participant (actor, step or machine) and then searching for a net model that can be a mapping of all these separate maps. With PrT nets, these mappings are inherently two-way.

The small model in Example 10 shows two PrT nets with lines denoting the mapping that can be used to derive their components from one another. One sees that these two nets are basically at the same level of specificity and that predicates and transitions can be used to map entire subnets in both directions. This possible structuring method suggests in itself the construction of separate but deeply linked nets where one net (the player) generates actual output events and another net (the interpreter) influences the parameters of such events.



(The directionality of the edges is ignored here.)

Printed in the *Proceedings of the 1982 International Computer Music Conference* and in Herbert Bruhn, Rolf Oerter, and Helmut Roessing, *Musikpsychologie: Ein Handbuch in Schlüsselbegriffen* (Munich: Verlag Urban und Schwarzenberg, 1985). Used by permission of the publisher.

EXAMPLE 10: PrT NET SHOWING MAPPING MORPHOLOGY BETWEEN TWO NETS

Just as with the state machines, we can remove control information and expand PrT nets into linear, timeline-like nets that more resemble familiar musical notation. In this case, transition conditions become simple time decisions (i.e., fire at t = 21, play one note). This allows one to define mappings between nets that remove or restore control and/or time information as was seen with state machines. The difference here is that this transformation can be described

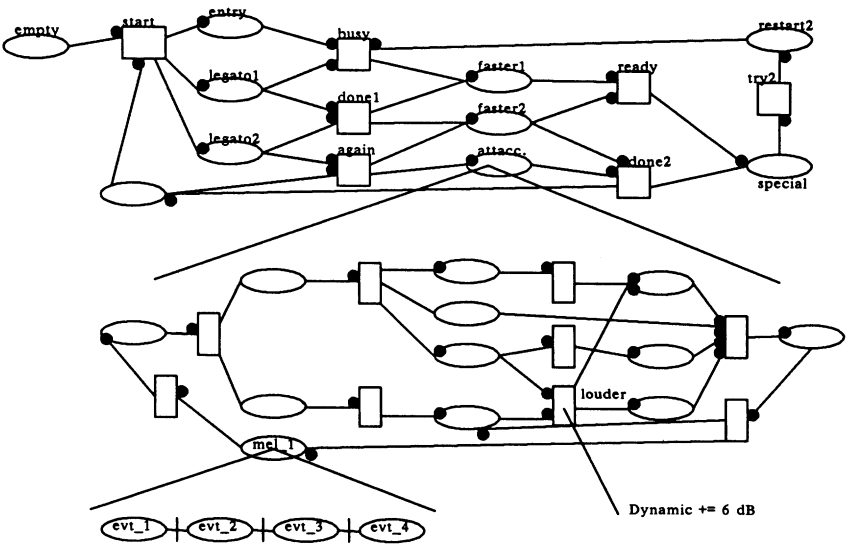
for PrT nets in a completely two-way manner using PrT net algebra (and some vector algebra).

As mentioned above, there are several possible representations or models for predicates and transitions in these nets. They can be seen as parts of a knowledge-based program as found in expert systems or automatic deduction programs or as mixtures of other process model types. The flexibility of these models leads one to think of basing a family of music notations on PrT nets with software tools to perform the expansion and mappings between varying levels of nets or other representations (state machines, trees or event lists).

Many very interesting possibilities arise when one thinks of designing a set of linked music notations that would allow a composer to develop a composition on several levels simultaneously, with a (software) system that could apply changes made to one representation to the other (related) notations.

The mappings that are possible with PrT or other nets or machine types also lead naturally to the consideration of mixing generator and modifier networks within a composition or mixing generator and modifier elements within one net. This means that a mapping between several nets can be defined as mentioned above, with separate nets for the different aspects of the event list output, or with separate nets for different notes or motives (time-sequenced nets).

A section of a partially marked PrT net that includes a mixture of generator and selector predicates is shown in Example 11. Here we can see that the predicate labeled *mel__1* at the middle net level directly determines a melody that is to be put in the simulation's output notelist while the transition labeled *louder* sets some global dynamic variable for all subsequent events.



EXAMPLE 11: MULTI-LEVEL PRT NET SHOWING TIME AND CONDITION LINKING

To produce music with such a net, one defines a marking for the net elements (predicates and transitions) and sets up some initial object state (loaded predicates and possible transitions). The simulation (whether by hand or by software) will involve transitions switching when their conditions can be fulfilled and the capacities of their input and output predicates allow the object passing defined by the transition.

The actual software implementation of such a modeling and simulation system, with graphical interaction and a comfortable user interface, is currently under way using some of the software techniques mentioned below and written using the Smalltalk-80 (TM of Xerox) programming environment (see Pope 1986a).

The intent is to offer the composer a collection of computer-based graphical editors for all of the notations described here and a set of tools for building complex models and describing relationships between different representations of their elements.

Software Implementations of Animated Musical Notations

What we are trying to show with these examples is the difference and improvement found when one moves away from basing musical programming on well-known software-determined techniques and towards building models of arbitrary temporal structures in a flexible (extensible) computer toolkit environment.

The development of more natural input/output (I/O) interfaces (front-ends) for music input languages (MILs) has been one of the main topics in computer music research. Among the important questions in this field are:

What defines the "instrument" in computer sound synthesis? (How do these elements influence the notation?)

the software synthesis method (additive synthesis, f.m., vocoders, . . .)

the user front-end and I/O devices (normal terminal, fancy graphics interfaces, conducting boxes, . . .)

the music software programming language (SCORE, cmusic, . . .)

the compositional model used for the score (note entry, non-deterministic, . . .)

the digital hardware used (synthesizer, computer, . . .)

What is the relevance of high-level models of processes relative to normal score languages?

How is the lack of a (human-readable) score for many newer compositions relevant to music analysis?

Is the lack of the possibility of further interpretation of "tape music" relevant?

The fields related to Artificial Intelligence (AI) and the cognitive sciences have provided several software advances in the handling of data-bases and knowledge-bases and in machine inference using expert systems (ES) technology. This has been especially beneficial for managing very large amounts of information and for displaying and manipulating multiple representations of data or knowledge structures.

Modern knowledge representation (KR) methods afford efficient and flexible access to such structures in any of several basic paradigms. The specific methods that can be used to implement the notations described here are: *Frames* (hierarchical knowledge structures), *Nets* (relation-based knowledge structures), and *Active Objects* (data-driven control structures).

The matters of inference and deduction in expert systems and their front-ends have grown into the subfields: inference engines, ES shells, production systems and knowledge-base management system (KBMS) tools. There is a wealthy and rapidly-growing literature and subculture in these areas.

Systems have been developed in the field of machine learning which are able to extract common properties from several examples in order to generalize wider principles and then to specify remaining deficiencies in a model or to interrogate a user on the basis of an evaluation of examples given in order to more precisely define a concept.

Some of the problems with directly applying these techniques to available music software or music theories are the complexity of representation structures for items that have many higher-level mappings and the difficulty of designing a good KR when the specification of the knowledge to be stored is very unclear at the onset of the system's programming (knowledge acquisition).

The above-mentioned state-oriented representations using state machines, control flow and data flow models can also be extended using KR techniques. There are several approaches to be found in the literature (see Loy 1985 or Fry 1984) that use structures of several parallel players, mini-experts or process groups for very flexible process control in music notation and/or composition assistance.

The computer music literature offers several examples of implementations of similar models in the form of computer software running on various systems. The possibilities of designing simulators that can be used to animate these models or defining real-time links to output devices will certainly be explored in the future in these areas.

Questions:

It seems fitting to close with a series of basic questions (of the type that are normally put at the start of this kind of paper). These considerations may aid in the classification of new notational and compositional techniques for new musical situations.

What defines the "instrument"? (see the discussion above)

What defines a "performance"? (a tape, a score, playing the tape, . . .)

What defines the "interpretation"? (score, composer's ideas, program, machine, . . .)

When is the composition process two or more steps, or when is the composer also the performer/interpreter/conductor?

What new performance/conducting situations are possible with compiled scores and some measure of real-time control?

What links between tree structures or sequences and timelines or machines are productive?

What is the relevance of the score? (none, for machines, for "read-along," for analysis, . . .)

How are traditional computer-music (instrument and/or score) programming languages (e.g., MUSIC5, SCORE, SSSP, etc.) extensible?

Conclusion

A general trend can be seen in the development direction of new high-level computer-music software and in modern notational tools. It can be said that future computer-music systems will need several types of notations, levels of notations and compositional mapping methods and that they must be readily extensible and customisable for each composer and each composition.

The most important factor in starting with all these considerations is to offer new musical directions to composers and/or performers with or without computer assistance. Among the most interesting aspects of the newly available interaction possibilities are the use of multi-player instruments and the resulting possible new conducting situations, the design of better musical automata and maybe even an extension of the range of use of traditional instruments.

Free, personalized and flexible music notations and new compositional strategies have been central points in new music theory and practice since the 1950s and applying new tools can only be expected to extend the range of musical possibilities available to composers in the future.

The full potential of computer-based music composition and editing systems is unleashed when these tools are linked to powerful synthesis and/or sound processing engines.

It is to be hoped that the rapid advances in computer-music and AI software will assert more conceptual influence on non-computer music and that composers of instrumental music can use modern methods and tools for music they write for live instrumental performance. The goal of encouraging and easing the use of new musical notations can best be met by a vigorous and open dialog between composers, software technicians, and those who are both.

As a last example, a picture of the screen of a bitmap graphics terminal running the prototype DoubleTalk music programming system is included (Example 12). This software is written in the Smalltalk-80 programming language. This system is currently under development and hopes to offer a flexible composer's assistant for process notation and music composition.

DoubleTalk offers graphical editors for PrT nets, state machines, sequence trees, and timelines (event lists) and will (hopefully) eventually be coupled to real-time synthesiser output as well as batch (non-real-time) sound synthesis software. The two companion papers to this one describe the DoubleTalk system and the rationale behind it in more detail.

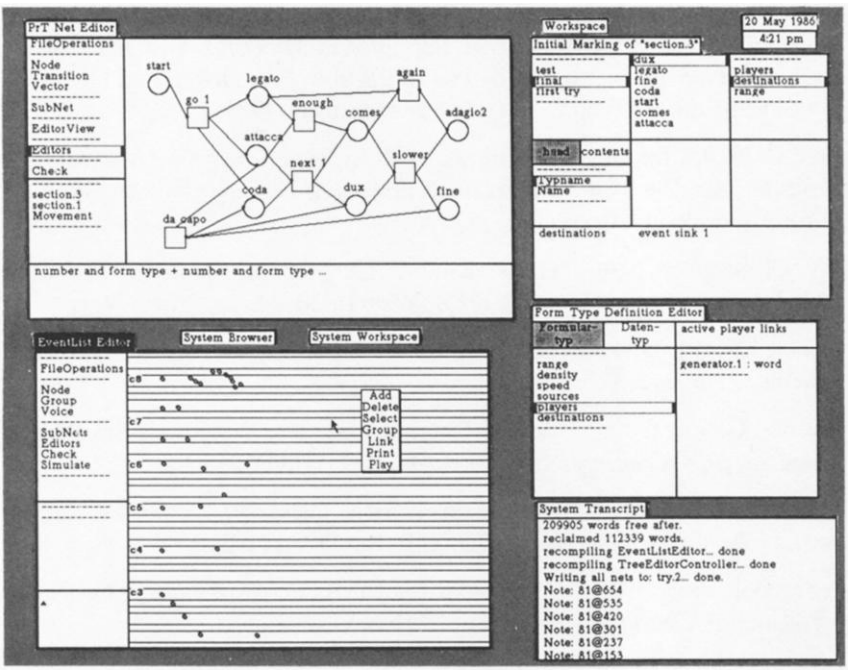
Acknowledgments

As mentioned in the introduction, this material grew out of discussions with a number of composers in Salzburg and Munich over the period of several years. I would like also to thank Curtis Roads and Tom Bagli, both of whom read and vastly improved earlier versions of this essay as well as all of the people at CMRS, IRCAM, SSSP, and the University of Dortmund who made the realisations mentioned here possible (and great fun).

(direct copy of the bitmap screen taken from a Cadmus 9000 running the Smalltalk-80 (TM of Xerox) programming environment)

The visible windows show several of the basic view types available within DoubleTalk, including:

PrT Net Editor	Edit Predicate–Transition Nets, subviews for menu operations, subnet menu and text input
PrT Net Form Type Editor	Define Object Types for PrT Net Simulations
PrT Net Marking Editor	Mark PrT Net Elements with predefined Forms and capacities for simulation and analysis
Event List Editor	Edit Event Lists generated by PrT Net Simulations, same subviews as in PrT Net Editor views
Smalltalk System Transcript	Scratchpad used by the system for messages, etc.
Smalltalk Browser (collapsed)	Used to enter new programs (objects and methods)
Smalltalk Workspaces (collapsed)	Used as scratchpads or for testing



EXAMPLE 12: SCREEN FROM THE DOUBLETALK COMPOSER'S ASSISTANT FRONT-END SOFTWARE

References

Abbott, Curtis. 1981. "The 4CED Program." *Computer Music Journal* 5, no. 1 (Spring): 13–33.

Buxton, William. 1978. "Design Issues in the Foundation of a Computer Based Tool for Music Composition." In *Technical Report CSRG-97*. Toronto: University of Toronto Press.

Craemer, D. 1978. "Netze aus Instanzen und Kanälen." In *Tagungsband: Methoden der Strukturierten Systementwicklung*. St. Augustin, Germany: G.M.D.

Decker, Shawn, and Gary Kendall. 1985. "A Unified Approach to the Editing of Time-Ordered Events." In *Proceedings of the International Computer Music Conference, 1985: Centre for the Arts, Simon Fraser University, Burnaby, B.C., Canada, August 19–22, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association.

Degli Antoni, Giovanni, and Geoffredo Haus. 1982. "Music and Causality." In *Proceedings of the International Computer Music Conference, 1982*. San Francisco, Calif.: Computer Music Association. Reprinted in *Musikpsychologie: Ein Handbuch in Schlüsselbegriffen*, edited by Herbert Bruhn, Rolf Oerter, and Helmut Roessing. Munich: Verlag Urban und Schwarzenberg. 1985.

Ellis, C. 1979. "Information Control Nets." In *Proceedings of the Association for Computing Machinery Conference on Simulation and Modelling of Computer Systems*. N.p.: Association for Computing Machinery.

Fry, Christopher. 1984. "Flavors Band: A Language for Specifying Musical Style." *Computer Music Journal* 8, no. 4 (Winter): 20–34.

Hatley, David J. 1984. "A Structured Analysis Method for Large, Real-Time Systems." *The Heap*, February.

Krasner, Glenn. 1980. "Machine Tongues VIII: The Design of a Smalltalk Music System." *Computer Music Journal* 4, no. 4 (Winter): 4–14.

Loy, D. Gareth. 1985. "Player Documentation." In *CARL Distribution Package*. La Jolla, Calif.: University of California, San Diego CME.

Pennycook, Bruce. 1985. "Computer Music Interfaces." *Association for Computing Machinery Computing Surveys* 17, no. 2 (June).

Petri, C. A. 1976. "General Net Theory." In *Proceedings of the IBM/University of Newcastle Seminar*. Newcastle: University of Newcastle. Reprinted in *GMD Bericht Nr. III*. Munich: Oldenbourg Verlag, 1979.

Pope, Stephen T. 1986a. "Software for Process Description, Modelling and

Description." In *Proceedings of the G.I. Workshop: Wissen and Wissenrepräsentation*. Munich: Münchner Arbeitskreis für künstliche Intelligenz.

_____ . 1986b. "Thoughts about Music Input Languages." In *Proceedings of the International Computer Music Conference, 1986*. Den Haag: Computer Music Association.

Reisig, W. 1982. *Petrinetze: Eine Einführung*. Berlin: Springer Verlag.

Roads, Curtis. 1985. "Research in Music and Artificial Intelligence." *Association for Computing Machinery Computing Surveys* 17, no. 2 (June).

Truax, Barry. 1977. "The POD System of Interactive Composition Programs." *Computer Music Journal* 1, no. 3 (Fall): 30–39.

Yavelow, Christopher. 1985. "Music Software for the Apple Macintosh." *Computer Music Journal* 9, no. 3 (Fall): 52–67.

Yourdon, Edward. 1981. *Structured System Methodologies*. New York: Yourdon Press.

Bibliography

SMALLTALK-80 literature

Goldberg, Adele, and David Robson. *Smalltalk-80: The Language and its Implementation*. Palo Alto: Addison Wesley, 1983.

Goldberg, Adele. *Smalltalk-80: The Interactive Programming Environment*. Palo Alto: Addison Wesley, 1983.

Heeg, Georg. *Porting the Berkeley Smalltalk-80 System to the Cadmus 9000*. Dortmund: Georg Heeg, Universität Dortmund, 1985.

Kay, Alan, and Adele Goldberg. *Smalltalk-72 Instruction Manual* (Report SSL 76–6). Palo Alto: Xerox PARC, 1976.

Krasner, Glenn, et al. *Smalltalk-80: Bits of History, Words of Advice*. Palo Alto: Addison Wesley, 1983.

The Smalltalk-80 Newsletter. Available from: Nanette Harter, ParcPlace Systems, Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94303.

Lecture Notes from the Smalltalk-80 Seminar. Available from: H. Ganzinger, G. Heeg Institut für Informatik V, University of Dortmund, P.O. Box 500500, 4600 Dortmund 50, F.R. Germany.

Alexander, J. *Exploratory Programming Using Smalltalk*. Beaverton, OR: Comp. Res. Lab., Tektronix Inc., 1985.

London, R., and Robert Duisberg "Animating Programs Using Smalltalk." *IEEE Computer*, August, 1985.

MOBY Project (Modellierung von BüroSystemen) Report, Lecture Notes and User Manual. MOBY Working Group, Institut für Informatik II, University of Dortmund, P.O. Box 500500, 4600 Dortmund 50, F.R. Germany.

XSIS Analyst-85 Documentation. Pasadena: Xerox Special Information Systems, 1985.

AI, EVENTS, AND PROCESSES

Dannenberg, Roger. "Arctic: A Functional Language for Real-Time Control." *ACM Symposium on LISP and F.P.* N.p.: Association for Computing Machinery, 1984.

Faleti, James, and Robert Wilensky. *The Implementation of PEARL, A Package for Efficient Access to Representations in LISP*. Berkeley, Calif.: University of California, Berkeley, Computer Science Division, 1982.

Fry, Christopher. "Flavors Band: A Language for Specifying Musical Style." *Computer Music Journal* 8, no. 4: (Winter 1984): 20–34.

Lieberman, Henry. "Thinking about Lots of Things at Once without getting Confused: Parallelism in ACT-1." *MIT AI Lab Memo No. 626*, 1981.

Messnick, Steven, and Kent Beck. *Active Variables in Smalltalk-80*. Beaverton, Ore.: Tektronix Inc., Computer Research Laboratory, 1985.

Feigenbaum, Edward, et al. "Signal-to-Symbol Transformation: HASP/SIAM Case Study." *AI Magazine* 3, 2 (1982).

Pope, Stephen T. "Software for Process Description, Modelling and Description." In *Proceedings of the GI Workshop: Wissen und Wissensrepräsentation*, Munich: Münchner Arbeitskreis für künstliche Intelligenz, 1986.

Rodet, Xavier, and Pierre Cointe. FORMES: Composition and Scheduling of Processes." *Computer Music Journal* 8, no. 3 (Fall 1984): 32–50.

Steels, Luc. *Programming with Objects using ORBIT*. Stamford, CT: Schlumberger Doll Research, 1981.

NETWORKS AND INFORMATION CONTROL

Claus, V. *Modellierung von Bürosystemen*. Dortmund: Spezialvorlesung, WS 84/85, Lehrstuhl Informatik II, Universität Dortmund, 1985.

Craemer, D. "Netze aus Instanzen und Kanälen." In *Tagungsband: Methoden der Strukturierten Systementwicklung*. St. Augustin, Germany: G.M.D., 1978.

Degli Antoni, Giovanni, and Geoffredo Haus. "Music and Causality." In *Proceedings of the International Computer Music Conference, 1982*. San Francisco, Calif.: Computer Music Association, 1982.

Ellis, C. "Information Control Nets." In *Proceedings of the Association for Computing Machinery Conference on Simulation and Modeling of Computer Systems*. N.p.: Association for Computing Machinery, 1979.

Genrich, H., and K. Lautenbach. "System Modeling with High-Level Petri Nets" *Theoretical Computer Science* 13 (1981).

Hatley, D. J. "A Structured Analysis Method for Large, Real-Time Systems." *The Heap*, February 1984.

InterProgram—Precedence Editor Notes—Blues Bulletin. Diemen, Holland: InterProgram B.V., 1985.

Peterson, J. L. *Petri Net Theory and the Modeling of Systems*. Englewood Cliff, N.J.: Prentice Hall, 1981.

Petri, C. A. "General Net Theory." In *Proceedings of the IBM/U. Newcastle Seminar, 1976*, reprinted in *GMD Bericht Nr. 111*. Munich: Oldenbourg Verlag, 1979.

Petri, C. A. "Kommunikationsdisziplinen." In *GMD Bericht Nr. 111*. Munich: Oldenbourg Verlag, 1979.

Richter, G., and R. Durchholz. "IML—Inscribed High-Level Petri Nets." In *Proceedings of the IFIP WG 8.1 Conference on Information Systems Design Methodologies*. N.p.: n.p., 1982.

Reisig, W. *Petrinetze: Eine Einführung*. Berlin: Springer Verlag, 1982.

Valk, R. "Self-Modifying Nets: A Natural Extension of Petri Nets." In *Colloquium on Automata, Languages and Programming*. Berlin: Springer Verlag, 1978.

Wasserman, Anthony, et al. *Transition Diagram Editor Description—Structured Development Tools*. San Francisco, CA: Interactive Development Tools, 1985.

Wißkirchen, Peter, et al. *Informationstechnik und Bürosysteme*. Stuttgart: Teubner Verlag, 1983.

_____. "Ein Rechnergestützter Bürosimulator auf der Basis von PrT-Netzen." *Angewandte Informatik* 5 (1984).

Yourdon, Edward. *Structured System Methodologies*. New York: Yourdon Press, 1981.

Proceedings of the Advanced Course on Petri Nets. St. Augustin, Germany: GMD, 1986.

MUSIC NOTATIONS AND COMPUTER INTERFACES

Baecker, Ronald. "Human-Computer Interactive Systems: A State-of-the-Art Review." In *Processing of Visible Language II*. Plenum, N.Y.: n.p. 1980.

Buxton, William. "Design Issues in the Foundation of a Computer-Based Tool for Music Composition." *Technical Report CSRG-97* Univ. of Toronto, October, 1978.

Banger, Colin, and Bruce W. Pennycook. "Gcomp: Graphics Control of Mixing and Processing." *Computer Music Journal* 7, no. 4 (Winter 1983): 33–39.

Brinkman, Alexander. "A Design for a Single-Pass Scanner for the DARMS Coding Language." In *Proceedings of the Rochester 1983 International Computer Music Conference*. San Francisco, Calif.: Computer Music Association, 1983.

Haynes, Stanley. "The Musician-Machine Interface in Digital Sound Synthesis." *Computer Music Journal* 4, no. 4 (Winter 1980): 23–44.

Kowalski, Michael J., and Andrew Glassner. "The N.Y.I.T. Digital Sound Editor." *Computer Music Journal* 6, no. 1 (Spring 1982): 66–73.

Lieberman, Henry. "Machine Tongues IX: Object-Oriented Programming." *Computer Music Journal* 6, no. 3 (Fall 1982): 8–21.

Mathews, Max, and F. R. Moore. "GROOVE: A Program to Compose, Store and Edit Functions of Time." *Communications of the Association for Computing Machinery* 13, no. 12 (December 1970).

Pennycook, Bruce W. "Music Languages and Preprocessors: A Tutorial." In *Proceedings of the Rochester 1983 International Computer Music Conference*. San Francisco, Calif.: Computer Music Association, 1983.

Roads, Curtis. "An Overview of Music Notations." In *Proceedings: Conference on Musical Grammars and Computer Analysis, Modena, 1982*. Milan: Oelschki, 1983.

COMPUTER MUSIC COMPOSITION SOFTWARE SYSTEMS

Abbott, Curtis. "The 4CED Program." *Computer Music Journal* 5, no. 1 (Spring 1981): 13–33.

_____ . *System Level Software for the LucasFilm ASP System*. N.p.: LucasFilm Ltd, 1982.

Buxton, William, et al. "The Structured Sound Synthesis Project (SSSP): An Introduction." *Technical Report CSRG-92* Univ. of Toronto, May, 1978.

Buxton, William, Richard Sniderman, William Reeves, Sanand Patel, and Ronald Baecker. "The Evolution of SSSP Score Editing Tools." *Computer Music Journal* 3, no. 4 (Winter 1979): 14–25, 60.

Buxton, William, Sanand Patel, William Reeves, and Ronald Baecker. "Scope in Interactive Score Editors." *Computer Music Journal* 5, no. 3 (1981): 50–56.

Buxton, William, Sanand Patel, William Reeves, and Ronald Baecker. "Objed and the Design of Timbral Resources" *Computer Music Journal* 6, no. 2 (1982): 32–44.

Chabade, Joel. "Interactive Composing: An Overview" *Computer Music Journal* 8, no. 1 (1984): 22–27.

Chafe, Chris, Bernard Mont-Reynaud, and Loren Rush. "Toward an Intelligent Editor for Digital Audio: Recognition of Musical Constructs" *Computer Music Journal* 6, no. 1 (1982): 30–41.

Cyphers, D. S. *SPIRE Programmer's Guide*. Cambridge: MIT Speech Group, 1983.

Decker, Shawn, Gary Kendall, et al. "A Modular Sound Synthesis Software Environment." In *Proceedings of the International Computer Music Conference, 1984*. Edited by William Buxton. San Francisco, Calif.: Computer Music Association, 1985.

_____. *Event List Library Manuals—libel.a Documentation*. Evanston: Northwestern University, Computer Music Studio, 1984.

_____. "A Unified Approach to the Editing of Time-Ordered Events." In *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association, 1985.

Goldberg, Theo. "The Preconfiguration of A Musical Composition: Model of a Computer Graphics Program." In *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association, 1985.

Greussay, Patrick, Jacques Arveilleur, Marc Battier, Chris Colere, Gilbert Dalmasso, Giusseppe Englert, and Didier Ronein. "Musical Software: Descriptions and Abstractions of Sound Generation and Mixing." *Computer Music Journal* 4, no. 3 (Fall 1980): 40–47.

Holtzman, S. R. "Using Generative Grammars for Music Composition." *Computer Music Journal* 5, no. 1 (Spring 1981): 51–64.

Jones, Kevin. "Compositional Applications of Stochastic Processes." *Computer Music Journal* 5, no. 2 (Summer 1981): 45–61.

Kaufmann, David. *An Introduction to the Speech Group LISP Machines*. Cambridge: MIT Speech Group. 1983.

Krasner, Glenn. "Machine Tongues VIII: The Design of a Smalltalk Music System." *Computer Music Journal* 4, no. 4 (Winter 1980): 4–14.

Lentczner, Mark. "SoundKit—A (Smalltalk) Sound Manipulator." In *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association, 1985.

Loy, D. Gareth. "Notes on the Implementation of MUSBOX: A Compiler for the Systems Concepts Digital Synthesizer." *Computer Music Journal* 5, no. 1 (Spring 1981): 34–50.

Loy, D. Gareth. "Player Documentation." In *CARL Distribution Package*. La Jolla, Calif.: University of California, San Diego CME, 1985.

_____. "Designing an Operating Environment for a Realtime Performance Processing Environment." In *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association, 1985.

_____. "Musicians make a Standard: The MIDI Phenomenon." *Computer Music Journal* 9, no. 4 (Winter 1985): 8–26.

Malouf, Frederick. "A System for Interactive Music Composition through Computer Graphics." In *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association.

Mathews, Max. "An Acoustical Compiler for Music and Psychological Stimuli." *Bell System Technical Journal* 40 (1961).

Maxwell, John T. "Mockingbird: An Interactive Composer's Aid." Master's thesis, MIT, January, 1981.

McNabb, Michael. "DreamSong: The Composition." *Computer Music Journal* 5, no. 4 (Winter 1981): 36–53.

Moore, F. Richard. "The CARL Computer Music Workstation—An Overview." In *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association, 1985.

Moorer, James A. "Synthesizers I have Known and Loved." *Computer Music Journal* 5, no. 1 (Spring 1981): 4–12.

Moorer, James A. "The LucasFilm Audio Signal Processor." *Computer Music Journal* 6, no. 3 (Fall 1982): 22–32.

Pennycook, Bruce. "Computer-Music Interfaces: A Survey." *Association for Computing Machinery Surveys* 17, no. 2 (June 1985).

Pope, Stephen T. "Introduction to the Music Shell." In *Proceedings of the International Computer Music Conference, 1982*. Venice: Computer Music Association, 1982.

_____. *Towards an Interpreter for Computer Music*. Salzburg: ComputerMusik Rechenzentrum Salzburg, 1983.

_____. "Thoughts about Music Input Languages." In *Proceedings of the International Computer Music Conference 1986*. Den Haag: Computer Music Association, 1986.

Roads, Curtis. "A Report on SPIRE: An Interactive Audio Processing Environment." *Computer Music Journal* 7, no. 2 (Summer 1983): 70–74.

Schottstaedt, Bill. "Pla: A Composer's Idea of a Language." *Computer Music Journal* 7, no. 1 (Spring 1983): 11–20.

Shipman, David. *Development of Speech Software on the MIT LISP Machine*. Cambridge: MIT Speech Group, 1982.

Smith, Leland. "Editing and Printing Music by Computer." *Journal of Music Theory* 17 (1973): 292–309.

_____. "SCORE—A Musician's Approach to Computer Music." *Journal of the Audio Engineering Society* 20, no. 1 (1974).

Snell, John. "The LucasFilm Real-Time Console for Recording and Performance of Computer Music." *Computer Music Journal* 6, no. 3 (Fall 1982): 33–45.

Truax, Barry. "The POD System of Interactive Composition Programs." *Computer Music Journal* 1, no. 3 (Fall 1977): 30–39.

_____. "The PODX System: Interactive Composition Software for the DMX-1000." *Computer Music Journal* 9, no. 1 (Spring 1985): 29–38.

Yavelow, Christopher. "Music Software for the Apple Macintosh." *Computer Music Journal* 9, no. 3 (Fall 1985): 52–67.

COMPUTER MUSIC AND ARTIFICAL INTELLIGENCE

Alphonce, Bo H. "Music Analysis by Computer: A Field for Theory Formation." *Computer Music Journal* 4, no. 2 (Summer 1980): 16–35.

Ashley, Richard D. "KSM: An Essay in Knowledge Representation in Music." *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association, 1985.

Balaban, Mira. "Foundations for AI Research of Western Tonal Music." *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association, 1985.

Cointe, Jean-Pierre, et al. "The Formes System: A Musical Application of Object-Oriented Concurrent Programming." In *Rapport IRCAM*. Paris: IRCAM, 1986.

Barlow, Clarence. "The Bus Journey to Parametron" *Die FeedBack Papers* 21-23 (1980): 1–124.

Damm, Klaus. *sgl - Struktur-Generator-Programm: Benutzerhandbuch*. Euskirchen Kuchenheim: n.p., 1984.

Duisberg, Robert. "On the Role of Affect in Artificial Intelligence and Music." *Perspectives of New Music* 23, no. 1 (Fall-Winter, 1984): 6–35.

Hiller, Lejaren, and Charles Ames. "Automated Composition: An Installation at the 1985 International Exhibition in Tsukuba, Japan." *Perspectives of New Music* 23, no. 2 (Spring–Summer, 1985): 196–215.

Holtzman, S. R. "Music as System." *Interface* 7 (1978): 173–87.

Laske, Otto. "A.I. Topics in Computer-Aided Composition: A Tutorial." *Proceedings of the Rochester 1983 International Computer Music Conference*, N.p.: n.p., 1983.

_____. "Keith: A Rule System for Making Music-Analytical Discoveries." *Proceedings of the Conference on Musical Grammars and Computer Analysis, Modena, 1982*. Milan: Oelschki, 1983.

Lerdahl, Fred, and R. Jackendoff. *A Generative Theory of Tonal Harmony*. Cambridge: MIT Press, 1983.

Mahling, Andreas. *Erweiterte Generative Grammatiken zur Definition und Beschreibung musikalischer Strukturen*. Stuttgart: Institut für Informatik, 1985.

Minsky, Marvin. "Music, Mind, and Meaning." *Computer Music Journal* 5, no. 3 (Fall 1981): 28–44.

Moorer, James A. "Music and Computer Composition." *Communications of the ACM* 15, no. 2 (1972).

Morris, S. "A Musical Knowledge-Based System using Active Objects." In *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association, 1985.

Rahn, John. "On Some Compositional Models of Music Theory." *Computer Music Journal* 4, no. 2 (Summer 1980): 66–72.

Roads, Curtis. "Grammars as Representations for Music." *Computer Music Journal* 3, no. 1 (Spring 1979): 48–55.

_____. "Artificial Intelligence and Music." *Computer Music Journal* 4, no. 2 (Spring 1980): 13–25.

_____. "Interview with Marvin Minsky." *Computer Music Journal* 4, no. 3 (Fall 1980): 25–39.

Roads, Curtis, and John Strawn, editors. *Foundations of Computer Music*. Cambridge: MIT Press, 1985.

Roads, Curtis. "Research in Music and Artificial Intelligence." *Association for Computing Machinery Computing Surveys* 17, no. 2 (June 1985).

Smoliar, Steven. "Process Structuring and Music Theory." *Journal of Music Theory* 18 (1974): 308–36.

Strawn, John. "Approximation and Syntactic Analysis of Amplitude and Frequency Functions for Digital Sound Synthesis." *Computer Music Journal* 4, no. 3 (Fall 1980): 3–24.

Sundberg, Johan, Anders Askenfelt, and Lars Frydén. "Musical Performance: A Synthesis-by-Rule Approach." *Computer Music Journal* 7, no. 1 (Spring 1983): 37–43.

Tenney, James, and Larry Polansky. "Temporal Gestalt Perception in Music." *Journal of Music Theory* 24 (1980): 205–41.

Thomas, Marilyn T. "Vivace: A Rule-Based AI System for Composition." In *Proceedings of the International Computer Music Conference, 1985*. Edited by Barry Truax. San Francisco, Calif.: Computer Music Association, 1985.

Wessel, David L. "Timbre as a Musical Control Structure." *Computer Music Journal* 3, no. 2 (Summer 1979): 45–52.

See also related bibliographies:

Smalltalk-80 Bibliographies in: *Cadmus 9000 Smalltalk-80 Introduction* (Universität Dortmund, Lehrstuhl Informatik V), and *Materials for AI: PCS BST-AI Group* (Munich: n.p., 1986).

UNCLE (UNIX CommonLISP Environment) Project AI Bibliographies in: *Proceedings of the Nordic UNIX Conference, Stockholm, October, 1985* (N.p.: n.p., n.d.), and *Materials for AI: PCS BST-AI Group* (Munich: n.p., 1986).