A Brief Survey of Music Representation Issues, Techniques, and Systems[1]

Roger B. Dannenberg
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
Email: dannenberg@cs.cmu.edu

1. Introduction

Music offers a challenging array of representation problems. As an art form, music is distinguished by the presence of many relationships that can be treated mathematically, including rhythm and harmony. There are also many non-mathematical elements such as tension, expectancy, and emotion. Music can contain symbolic or structural relationships existing within and between the dimensions of pitch, time, timbre, harmony, tempo, rhythm, phrasing, and articulation. A further source of complexity is that ‘‘music’’ can mean printed notation, performance (instrument control) information, or resulting sounds. Finally, music evolves with every new composition. There can be no ‘‘true’’ representation just as there can be no closed definition of music. These elements combine to make music representation a rich field of study.

Computers allow (and require) a formal approach to the study of music representation. Computer programs demand that every detail of a representation be precisely specified, and the resulting precision allows experimentation and testing of new representations. The knowledge we gain from these experiments is useful in music theory and in music understanding by computer. Computers also allow the representation of dynamic, responsive, or interactive compositions, a concept that was hard to imagine before computing. Better representations also lead to more powerful human/computer interfaces and tools.

This is an overview of some of the issues and techniques that arise in studies of music representation. No attempt has been made to be exhaustive, so the reader is referred to the bibliography and the other articles in this issue for more information. In particular, I will omit any discussion of audio representations and transforms used in signal processing [DePoli 91], although this is an important part of music representation.

[1]Published as: Dannenberg, ‘‘Music Representation Issues, Techniques, and Systems,’’ ***Computer Music Journal***, 17(3), pp. 20-30.

2. Levels of Representation

Musicians deal with many levels of abstraction in music. If a conductor says, ‘‘play the downbeat with more conviction,’’ he or she is referencing music structure (a downbeat) and emotional content in the same sentence. It is convenient to think of musical representations at different levels, ranging from the highly symbolic and abstract level denoted by printed music to the non-symbolic and concrete level of an audio signal. Performance information is an intermediate level. We must consider these levels because there is rarely a unique conversion between any two of each of them. Each level contains at least some information not available in other levels. In general, there is great interest and value in performing automatic (partial) conversions between levels [Katayose 89], such as in optical music recognition or music transcription.

3. Hierarchy and Structure

Early computer music systems, especially those intended for music synthesis, represented music as a simple sequence of notes. The Music V score language [Mathews 69] is a good example. This approach is simple, but it makes it difficult to encode structural relationships between notes. For example, applying an amplitude envelope to a group of notes is a tricky operation in Music V and its successors.

MIDI is similar in that it has no mechanisms for describing new structural relationships. However, MIDI has a number of predefined structures. There are 16 channels, which effectively form 16 groups of notes. Each group has a set of controllers such as volume and pitch-bend. This gives MIDI a limited 2-level structure.

Many researchers have investigated hierarchical representations. For example, Buxton [Buxton 85a] describes a system where ‘‘events’’ can be notes or sequences of events, allowing arbitrarily nested structures. One advantage of hierarchically structured descriptions of music is that transformations such as tempo or pitch can be applied to aggregates of musical objects. In general, hierarchy is a way of representing structure, and it should be no surprise that many composition languages support the notion of hierarchy.

A single hierarchy scheme is still limiting because music often contains multiple hierarchies. Consider beams and phrase marks found in music notation. Notes can have several beams, and beams are often broken to help subdivide rhythms, so a multilevel beam hierarchy arises naturally. Phrase markings, ties, and slurs form another multilevel hierarchy that is completely separate from beaming. Consider some other possible hierarchies: voices (orchestra, strings, violins, 1st violins, solo violin), sections (movement, section, measure), phrases (which may cut across sections), and chords, all of which are ways of grouping and structuring music. A single hierarchy system is inadequate to represent all these concepts at the same time.

Brinkman [Brinkman 85] and Dannenberg [Dannenberg 90] describe representations that support multiple hierarchies through named links relating musical events or objects, and through explicit objects that represent instances of hierarchies. The NeXT Music Kit and HyTime support a more indirect approach where events may be given ‘‘tags’’ to indicate grouping. For example, all notes under a phrase marking may receive the tag ‘‘phrase257.’’

4. Extensibility

If musical information was well-understood and fixed, then music representation would be a much simpler problem. In reality, we do not know all there is to know, and the game is constantly changing. For both of these reasons, it is important that music representations allow extensions to support new concepts and structures.

An example of extensibility is seen in Music V. Here, the orchestra language allows the composer to assemble unit generators into a collection of synthesis algorithms for synthesizing notes. The most interesting aspect of this arrangement is that every synthesis algorithm determines both the number and meaning of control parameters. The score language allows each note to have a corresponding number of parameters. This is in contrast to MIDI, where the parameters of voice, pitch, and velocity are fixed. MIDI *could* be extended fairly simply by designating some MIDI Control Change messages to set additional parameters. For example, Control Numbers 20 through 31 might be dedicated as note parameters P1 through P12 which would then be sent as a prefix to each note that uses extra parameters.

Music V influenced many other systems, and a frequent modification has been the use of named and typed properties. Examples include event-lists [Decker 84], item-lists [Dannenberg 90], PLA [Schottstaedt 83], SMDL [Newcomb 91], and MODE [Pope 91], to name just a few. In these systems, note parameters are represented as a set of properties, usually consisting of name/value pairs, for example, [Pitch: C4, Instrument: Violin, Duration: 1 Quarter, Dynamic: Forte].

One of the disadvantages of extensible structures such as these is the difficulty of providing *semantic* extensions to deal with additional parameters. Many of the most flexible representations systems are part of a programming environment, and it is up to the composer to implement interpretations for new parameters by programming.

If programming is necessary, one would like at least to write one global description of what it means to, say, interpret a Vibrato property, rather than modifying many instrument descriptions. Multiple inheritance schemes in object-oriented and frame-based systems have the potential to support such extensions, but in practice, real systems often support certain classes of extensions and break down on others. Designing representations that can be extended with both raw information and semantic interpretations of the data is still an area of research. [Bocker 88]

Even fairly closed representations such as MIDI are full of interpretation problems. For example, MIDI specifies how to encode key velocity and pitch bend, but it does not specify quantitatively how these parameters should affect sounds. Should key velocity translate into decibels? Should it map to some subjective loudness measure? [Martens 85] Should it map uniformly to the dynamic range of a corresponding acoustic instrument? These are all conceivably valid interpretations. The current state of the art is that manufacturers provide a set of suggestive defaults, and composers either adjust synthesizer patches to obtain uniform responses or they adjust their MIDI data separately for each voice to get the desired results.

5. Pitch

The seemingly simple concept of pitch is in practice fairly complex. This is because pitch exists as an acoustic property (repetition rate), a psychological percept (perceived pitch), and as an abstract symbolic entity relating to intervals and keys. Psychologists have measured the relationship between the perceived pitch or interval size and the fundamental frequency of a tone. The relationship depends upon many variables. [Krumhansl 91] A consequence is that a concept as basic as the octave can have many meanings.

At the symbolic level, pitches participate in mathematical structures [Balzano 80, Shepard 82]. To name a pitch, we need to specify a scale, octave, scale step, and alterations, e.g. the pitch C# in the fourth octave of an equal-tempered scale based on A440. Descriptions like this assume scales establish finite sets of pitch choices. In practice, performers vary pitch for a variety of reasons: Soloists often play a little sharp for added salience, performers may retune dynamically to eliminate beating, and ensembles exhibit pitch variations producing chorus effects.

These are more problems of models, or deciding what to represent, rather than how to represent it. I mention these problems here to indicate the richness required of any general pitch representation.

6. Tempo, Beat, Duration, and Time

The interaction between real time, measured in seconds, and metrical time, measured in beats, is frequently addressed in music representation schemes. Abstractly, there is a mathematical function that maps beat numbers to time and an inverse function that maps time to beats. (Some special mathematical care must be taken to allow for music that momentarily stops, instantaneously skips ahead, or is allowed to jump backwards, but we will ignore these details.)

One practical representation of the beat-to-time function is the set of times of individual beats. For example, a performer can tap beats in real time and a computer can record the time of each tap. This produces a finitely sampled representation of the continuous mapping from beats to time, which can be interpolated to obtain intermediate values. MIDI Clocks, which are transmitted at 24 clocks per quarter note, are an example of this approach.

The Mockingbird score editor [Maxwell 84] interface also supported this idea. The composer would play a score, producing a piano-roll notation. Then, downbeats were indicated by clicking with a mouse at the proper locations in the piano roll score. Given this beat information, Mockingbird could then derive a symbolic score.

Another popular technique is the ‘‘tempo curve’’ [Rogers 80, Jaffe 85], a function from beat number to tempo, where tempo is the instantaneous ratio of beats per second, or equivalently, the first derivative of the function from time to beats. (A related function is tempo as a function of time rather than beats.) The tempo curve is a nice abstraction for mathematical specification. For example, logarithmic tempo curves have been found to produce smooth tempo changes that are musically plausible (at least better than linear).

There are some interesting numerical precision issues created by considerations of tempo and rhythm. The first is that rounding errors in note durations must not cause noticeable errors. For example, suppose a system represents duration in milliseconds. If a quarter note has a duration of 1000ms, then eighth-triplets will have a duration (after rounding) of 333ms. A series of 30 triplets played against 10 quarters will be skewed by 10ms. Some early computer music systems, undoubtedly feeling the pressure of memory limitations, developed encodings based on rational arithmetic to avoid round-off error altogether. This proved not to solve all the numerical problems.

Imagine now a system for film scoring where absolute timing is critical. Even if rational arithmetic is used to avoid round-off error in computing beats, the conversion from beats to seconds is subject to round-off error. If we arbitrarily tolerate 1ms of error per hour, then we need a precision of at least 22 bits, and more if many small durations are summed. Even 32-bit floating point numbers are not suitable for such calculations.

One of the advantages of computers is that they can perform calculations and manipulations on musical data. In the case of tempo, a common task is to fill a certain amount of time with a certain number of beats. Extensions of this idea include synchronizing certain beats with actions on film or video and manipulating multiple changing tempi while still achieving synchronization at certain points. This is essentially a constraint satisfaction problem, yet few representations for time and tempo can express constraints.

Before concluding this discussion of tempo and time, we should observe that mathematically elegant representations do not necessarily guarantee musical validity. Tempo transformations make mathematical sense, but do not always produce musical results. [Desain 91a] Research is needed to find transformations that retain musical nuance, and it has been suggested that explicit structure is essential [Desain 91b, Bilmes 92]. In the programming language area, I have suggested that transformations can be encapsulated into *behavioral abstractions* that support transformations [Dannenberg 89], and similar notions have been proposed by [Desain 92] and [Scaletti 92]. For example, the operation to stretch a trill is much different than the operation to stretch a melody. Here again, good representations can support the goal of musical transformations.

7. Timbre

With many aspects of music, we know what to represent, and the issue is how to represent it. With timbre, we are still learning what to represent. My explanation is that, starting with sound, we picked out the two things we understood, pitch and amplitude, and called everything else timbre. So timbre is by definition that which we cannot explain. As aspects of timbre are isolated and understood, such as spatial location and reverberation, these components come to be regarded separately, leaving timbre as impenetrable as ever.

Taking a less cynical view, real progress has been made toward timbre representation. The classic studies by Wessel [Wessel 85] and Grey [Grey 75] used multidimensional scaling and refined the notion of timbre space. While these studies represented timbre in terms of perceptual dimensions, others have represented timbre in terms of control dimensions, such as the set of parameters needed for a particular synthesis algorithm.

The current practice usually represents timbre by name or number. Real timbre space has so many dimensions that it is often preferable to locate and name interesting points in the space. MIDI Program Change messages and the concept of ‘‘instrument’’ found in many software synthesis systems are examples of this approach.

8. Music Notation

Music notation is a rich source of representational problems. One of the key problems is that music notation is not just a mechanical transformation of performance information. Performance nuance is lost going from performance to notation, and symbolic structure is lost in the translation from notation to performance. There is evidence of strong relationships between structure and performance nuance [Palmer 89, Clark 91]), but it seems unlikely that the relationships are strong enough to guarantee lossless transformations.

It seems that music notation rules are made to be broken. Donald Byrd's thesis [Byrd 84] contains an excellent discussion of notational variations and the impossibility of automatic music notation layout. Especially interesting is the fact that Byrd's examples, numbering about 100, all come from established, traditional composers and publishers. Two examples are the Henle edition of Chopin's *Nocturne*, Op. 15, No. 2, where one notehead serves both as a regular and a triplet 16th, and the Peters/Sauer edition of Brahm's *Capriccio*, Op. 76, No. 1, where a written dotted half note has an actual duration of eleven 16ths!

Another issue is that notation is visual and leaves open many layout choices. Thus, notation is partly a graphics design task, and manual layout is necessary. Representations for music notation usually include the representation of musical structure, such as key and time signatures, bar lines, beams, slurs, etc., and also graphical information such as staff position, stem direction, and graphical positioning. In many cases, layout information is implied by musical structure, but manual overrides are necessary.

The need to represent visual layout information and the fact that some of this information cannot be generated automatically has important implications for music notation systems. One is that copying parts from a score cannot be a fully automatic process, a problem that commercial notation systems have largely ignored. Most music notation programs allow the user to convert a score file into a set of parts files. However, complex scores will inevitably have many details that must be manually specified for each part, including page breaks, cues, and other notation that would not appear in the score. A problem occurs if changes are made to the score after parts have been manually corrected. Either the changes must be manually transferred to each part, or alternatively, the parts can be regenerated automatically. In the first case, there is no support for consistency between the score and the parts, and in the second case, the manual adjustments to each part are lost, and must be redone by hand.

A solution to this problem with parts is to represent parts as *views* on the score [Dannenberg 86a]. A *view* of a data structure contains a subset of the information in the data structure and sometimes provides alternate or additional data to that in the data structure. The idea is to keep shared data in one place so that a change in the score will automatically be propagated to the parts, and part-specific layout information can be maintained for each part (view).

Views have many other potential applications. Alternative notations for a single piece of music can be represented as views [Buxton 85b]. Even repeated sections of music might be represented as views. For example, it is common notational practice to indicate that a section of music is to be repeated. Often, each repetition has its own ending, or notation such as ‘‘tacet the first time.'' This is essentially a view mechanism in operation; each repetition shares most of the underlying information, but there are a few local alterations. If repeats are to be notated once but performed twice, views offer a mechanism for representing performance variations between the first and second repeat.

Carrying this to an extreme, we can imagine views as a general mechanism to represent structure in music. Imagine a representation where motives are represented only once, and each occurrence is some kind of view, perhaps with local alterations and transformations, of the motive. Such a scheme would quickly lead to many interesting problems. If a note in a view is edited, and then the original note in the motive is deleted, should the view's note be deleted as well? Can the user control such decisions? Can views be nested? These issues have much in common with representational schemes proposed for artificial intelligence, programming languages, and databases.

Continuing with other notational issues, one important feature of notation is that events are represented left to right in increasing time order, but the position is not in exact proportion to time or to beat number. An interesting situation arises when multiple tempi are present simultaneously, especially when the tempi are not related by simple fractions such as a 6/8 measure in the time of a 2/4 measure. In the more complicated situations, beat and tempo information must be combined to form absolute time, which then becomes the basis for left-to-right layout. I know of no music notation systems that can even represent this situation, much less perform reasonable layout. A similar situation would arise if absolute time notations for film, animation, or tape were to be graphically aligned with conventional music notation.

Composers would like to notate not only conventional notation but new graphical notations as well. In some ways, any good CAD or graphics package could support new notation, but it would be nice to have the graphics closely tied to underlying musical structures. Graphical editing should have a corresponding effect on musical parameters which might then control a music synthesizer. An interesting proposal along these lines was made by [Oppenheim 87], and examples of (non-editable) graphics representations can be seen in [Dannenberg 91a], [Brinkman 91], [Waters 90], and [Schottstaedt 83].

Another approach to extensible notation is found in [Assayag 86], a scheme in which an elaborate PostScript library was developed to assist in producing complex musical graphics. The library manages contraints among connected objects so that, for example, beams can be made to terminate at the end of a particular stem and other stems can be made to touch the beam but not go beyond it. The MusScribe notation system [Hamel 88] also abandons the issue of maintaining consistency between graphical notation and music structure by providing a graphical editor that is optimized for music notation. For example, noteheads snap to staff lines and spaces, but horizontal positioning is set manually.

9. Continuous and Discrete Data

Music information can be classified as either *continuous* or *discrete.* Continuous information changes over time (or perhaps as a function of other variables) and is typically represented by digital sampling, by splines (including piece-wise linear functions), or by arbitrary mathematical functions. In contrast to continuous data that fills time intervals, discrete information usually represents events at a point in time. A MIDI NoteOn event is an example. It is sometimes advantageous to represent discrete events using time intervals rather than points. [Honing 92] There is a natural correspondence to musical notes, which have duration as well as starting time. Also, intervals and sequences of intervals can be appended.

Continuous information is found in the signals of Music V and also in the Groove system [Mathews 70]. Groove was a real-time, multichannel, continuous information recorder and manipulator, and is highly recommended for further study.

Music representation systems have generally had a difficult time integrating continuous and discrete data. For example, many modern sequencers support smooth changes in controls such as volume, but functions of time are not first class entities that can be combined with mathematical operations. Another issue is the use of continuous data as parameters to discrete events. If a continuous function is to be used as a pitch contour, should each note sample the function to obtain a constant pitch, or should pitch vary over the course of the note? Finding general representations that incorporate continuous and discrete data is still an active area of research.

10. Declarative and Procedural Representations

Most of the representations discussed so far are encodings of static information. In contrast, computer programs are often used to encode dynamic behavior or as an alternate representation of static data. For the most part, visual/graphical editors, such as sequencers and notation editors, manipulate static data. Exceptions include Max [Puckette 91], Kyma [Scaletti 91, Scaletti 89], and numerous patch editors, which describe dynamic behavior.

While visual programming is still in its infancy, textual programming languages have a rich history, and the importance of time in music has led to a number of language innovations. A full treatment of languages for computer music cannot be given here, but we will introduce three very different language approaches to time.

FORMES [Rodet 84] is an object-oriented language in which objects represent behaviors. Each object maintains a start-time, a duration, and an operation to be performed at each time step. Objects are organized into tree structures and each object is activated at each time step to provide concurrency.

Arctic [Dannenberg 86b] is a functional language, where values are functions of time rather than simple scalars. Concurrency arises naturally whenever multiple values are computed because values can span overlapping intervals of time. Arctic-like semantics are found in Canon [Dannenberg 89], a composition system for generating MIDI data, and in Fugue [Dannenberg 92a] and Nyquist [Dannenberg 92b], systems for sound synthesis. GTF [Desain 92] is a proposal for linking control functions to attributes of discrete events. GTF shares Arctic’s functional programming orientation.

Formula [Anderson 86] is a procedural and process-oriented language. It uses the sequential execution of statements to represent sequences, notes and functions of time. Formula uses processes to obtain concurrent behavior. Of particular interest in Formula are its techniques for real-time scheduling and its support for nested time maps.

HMSL [Polansky 90] emphasizes hierarchical *morphologies*, or *morphs*. The basic morph is a multidimensional array of data; higher-level morphs can contain collections of other morphs. Morphs can provide a declarative, data-intensive representation of structure, but this data can be operated upon and interpreted by programmed procedures. HMSL emphasizes graphical interfaces to morphological data and operations.

11. Resources, Instances and Streams

There is an interesting distinction between musical entities that are freestanding and those that serve as parameters or updates to some other entity. Often, an entity has both characteristics. Take a note as an example. Some compositions and also languages like Music V treat notes as independent. In Music V, every note produces a copy or instance of a synthesis computation that is independent of other notes. MIDI synthesizers in poly mode also support this *instance* model.

In contrast, notes can also be viewed as updates or control information to some shared resource. For example, in orchestration one must be aware that a clarinet can only play one note at a time, and the line is as important as individual notes. Thus, notes are not independent instances of clarinet tones, but updates to a shared clarinet *resource*. This is the basis for the resource-instance model [Dannenberg 91b].

The resource-instance model, in which every entity is handled by some resource, helps to clarify and explain music representations. For example, in MIDI poly mode it is not specified whether a NoteOn message is an update to the channel (in which case two notes of the same pitch are possible) or whether a NoteOn is an update to a key number (in which case two NoteOn commands of the same pitch cause the note to be retriggered).

It is often convenient to think of a succession of events as a single entity, which we will call a stream. Streams are subject to various operations such as selection of events that satisfy some property, transformations such as transposition, and time deformation. Music Logo [Orlarey 86], Teitelbaum’s performance system [Teitelbaum 85], MAX [Puckette 91] and many sequencers (especially Bars and Pipes [Fey 89]) use this notation of streams.

A related concept is the sequence, where components are indexed by ordinal position rather than time. We speak of the seventh element rather than an element at time 3.2 seconds. Gary Nelson’s MPL [Nelson 77] used APL as a basis for a sequence-oriented compositional environment. The tone row is a sequence of 12 pitches. PLA [Schottstaedt 83] and Common Music [Taube 89] provide a number of sequence generators and use the sequence abstraction heavily.

12. Protocols and Coding

Once a music representation is adopted, issues of transmitting and storing the representation arise. Transmission, especially in real time, raises questions of network protocols, the conventions by which information is transmitted and received. Storage raises the question of coding, or how the abstract information is converted into specific bit patterns.

MIDI is the most prevalent protocol for the real-time transmission of music information, but it has many weaknesses. MIDI contains no mechanisms for: (1) flow control, which would eliminate receive buffer overflow; (2) (forward) error correction, which would enable receivers to detect errors and reconstruct garbled data; or (3) (backward) error recovery, which would provide a mechanism such as retransmission of lost or garbled data. Another limitation of MIDI is that there is no standard way to determine what devices are accessible to query their status, or to reserve resources. MIDI is based on the transmission of incremental state changes, which means the current state (parameter settings) of a synthesizer depends upon its entire message history. If a synthesizer is reinitialized during a performance, there is no way to recover the proper state. Finally, MIDI timing is not explicit. As MIDI data is processed, filtered, and merged, timing distortions occur. There is no way to encode timing specifications or timing distortions so that the intended timing can be reconstructed.

Many of the problems of MIDI are probably justified by economics. In the future, multigigabit network technology will completely change the set of assumptions on which MIDI is based and lead to very different protocols.

In the area of representation coding, two issues are human readability and the encoding of references. Codings can be made ‘‘human readable’’ by using ASCII text [Ames 85], mnemonic labels, and decimal numbers. Alternatively, codings can be optimized for space efficiency by using binary numbers throughout. References (i.e. pointers) are always problematic when data is encoded for storage in a linear file. A standard technique is to give each entity a unique name, which is then used to encode references to the entity. References tend to make codings harder to work with, but more flexible.

13. Other Issues

Representations for music databases must make searching efficient. At issue is what kinds of searches are allowed, how search queries are represented, and what kinds of indexes and other data structures can make searching efficient. Brad Rubenstein discusses extensions to a relational database to support music information storage and retrieval [Rubenstein 87].

Research in neural nets for music [Lischka 87, Todd 91] has led to the consideration of representations where information is distributed over collections of artificial neurons rather than stored in a discrete data structure. An important issue in this work is how data is presented to neural nets. For example, pitch can be specified in terms of frequency, pitch-class plus octave, or the set of triads in which the pitch participates [Hild 92]. Time in neural networks has been represented by using time-dependent cells [Scarborough 89], by successive iterations of a network with feedback [Jordan 86], and by spacial distribution [Sejnowski 86].

Cognitive studies such as Simon and Sumner [Simon 68] have investigated how humans represent musical information. Increasingly, music theory is used as a basis for hypothesis forming [Krumhansl 91], and it seems that perceptual studies are important for new representations and to validate old ones. The issue then is how can representations incorporate results from music psychology research.

14. More Information

Many music representation issues are special cases of more general problems addressed by computer science, so a good computer science background is important. Especially important is a knowledge of fundamental data structures such as lists, trees, and graphs, and algorithms for manipulating them [Horowitz 76]. Beyond this, the subfields of artificial intelligence and database systems are good sources of information on representation. Brachman and Levesque edited a collection of important papers on knowledge representation [Brachman 85] that have much relevance to music representation. Date’s book on database systems [Date 91] offers a wealth of material on the information organization, representation and access.

Almost all computer music research touches upon some aspect of representation, so a good way to learn more is simply to read back issues of Computer Music Journal and Proceedings of the International Computer Music Conferences, available from MIT Press and the International Computer Music Association, respectively. Many specific references have already been made. Honing’s article on time and structure in music [Honing 92] is especially recommended.

Music representation issues are often discussed informally and sporadically via electronic mail. Readers are invited to join an online discussion by sending electronic mail to the author at dannenberg@cs.cmu.edu.

15. Conclusion

Music is a fertile field for the study of representations. Music contains complex structures of many interrelated dimensions, including time. Music, music representations, and music theory are all evolving together, creating a continuous flow of new challenges. I have attempted to present some of the current issues and research directions in music representation. Comments and further discussion are welcome.

16. Acknowledgments

The author wishes to thank Stephen Pope, Carla Scaletti, and especially Henkjan Honing for their comments on an earlier draft.

References

[Ames 85] Ames, C. The ASHTON Score-Transcription Utility. *Interface* 14:165-173, 1985.

[Anderson 86] Anderson, D. P. and R. Kuivila. Accurately Timed Generation of Discrete Musical Events. *Computer Music Journal* 10(3):48-56, Fall, 1986.

[Assayag 86] Assayag, G., and D. Timis. A ToolBox for Music Notation. In P. Berg (editor), *Proceedings of the International Computer Music Conference 1986*, pages 173-178. International Computer Music Association, 1986.

[Balzano 80] Balzano, G. J. The Group-theoretic Description of 12-Fold and Microtonal Pitch Systems. *Computer Music Journal* 4(4):66-84, 1980.

[Bilmes 92] Bilmes, J. A Model for Musical Rhythm. In *ICMC Proceedings*, pages 207-210. Computer Music Association, San Francisco, 1992.

[Bocker 88] Bocker, H.-D. and A. Mahling. What's in a Note? In C. Lischka and J. Fritsch (editor), *Proceedings of the 14th International Computer Music Conference*, pages 166-174. International Computer Music Association, 1988.

[Brachman 85] Brachman, R. J., and H. J. Levesque, eds. *Readings in Knowledge Representation.* M. Kaufmann, Los Altos, Calif., 1985.

[Brinkman 85] Brinkman, A. A Data Structure for Computer Analysis of Musical Scores. In *Proceedings of the 1984 International Computer Music Conference*, pages 233-242. Computer Music Association, 1985.

[Brinkman 91] Brinkman, A. Computer-Graphic Tools for Music Analysis. In B. Alphonse and B. Pennycook (editor), *ICMC Montreal 1991 Proceedings*, pages 53-56. Computer Music Association, San Francisco, 1991.

[Buxton 85a] Buxton, W., W. Reeves, R. Baecker, and L. Mezei. The Use of Hierarchy and Instance in a Data Structure for Computer Music. *Foundations of Computer Music.* In C. Roads and J. Strawn, MIT Press, 1985, pages 443-466.

[Buxton 85b] Buxton, W., R. Sniderman, W. Reeves, R. Patel, and R. Baecker. The Evolution of the SSSP Score-Editing Tools. *Foundations of Computer Music.* In C. Roads and J. Strawn, MIT Press, 1985, pages 376-402.

[Byrd 84] Byrd, Donald. *Music Notation by Computer*. PhD thesis, Computer Science Department, Indiana University, 1984. Ann Arbor: University Microfilms (order no. DA8506091).

[Clark 91] Clark, E. F. Expression and communication in musical performance. *Wenner-Gren International Symposium Series, Vol. 59. Music, Language, Speech and Brain.* In J. Sundberg, L. Nord, and R. Carlson, Macmillan, London, 1991, pages 184-193.

[Dannenberg 86a] Dannenberg, R. A Structure for Representing, Displaying, and Editing Music. In P. Berg (editor), *Proceedings of the International Computer Music Conference 1986*, pages 153-160. International Computer Music Association, 1986.

[Dannenberg 86b] Dannenberg, R. B., P. McAvinney, and D. Rubine. Arctic: A Functional Language for Real-Time Systems. *Computer Music Journal* 10(4):67-78, Winter, 1986.

[Dannenberg 89] Dannenberg, R. B. The Canon Score Language. *Computer Music Journal* 13(1):47-56, Spring, 1989.

[Dannenberg 90] Dannenberg, Roger B. A Structure for Efficient Update, Incremental Redisplay and Undo in Display-Oriented Editors. *Software: Practice and Experience* 20(2):109-132, February, 1990.

[Dannenberg 91a] Dannenberg, R. B., C. L. Fraley, and P. Velikonja. Fugue: A Functional Language for Sound Synthesis. *Computer* 24(7):36-42, July, 1991.

[Dannenberg 91b] Dannenberg, R. B., D. Rubine, T. Neuendorffer. The Resource-Instance Model of Music Representation. In B. Alphonse and B. Pennycook (editor), *ICMC Montreal 1991 Proceedings*, pages 428-432. International Computer Music Association, San Francisco, 1991.

[Dannenberg 92a] Dannenberg, R. B., C. L. Fraley, and P. Velikonja. A Functional Language for Sound Synthesis with Behavioral Abstraction and Lazy Evaluation. *Readings in Computer-Generated Music.* In Denis Baggi, IEEE Computer Society Press, Los Alamitos, CA, 1992.

[Dannenberg 92b] Dannenberg, R. B. Real-Time Software Synthesis on Superscalar Architectures. In *Proceedings of the 1992 ICMC*, pages 174-177. International Computer Music Association, San Francisco, 1992.

[Date 91] Date, C. J. *An Introduction to Database Systems.* Addison-Wesley, Reading, Mass., 1991.

[Decker 84] Decker, S. and G. Kendall. A Modular Approach to Sound Synthesis Software. In W. Buxton (editor), *Proceedings of the International Computer Music Conference 1984*, pages 243-250. International Computer Music Association, 1984.

[DePoli 91] G. De Poli, A. Piccialli, and C. Roads (editor). *Representations of Musical Signals.* MIT Press, Cambridge, Mass., 1991.

[Desain 91a] Desain, P. and H. Honing. Tempo Curves Considered Harmful. In B. Alphonce and B. Pennycook (editor), *ICMC Montreal 1991 Proceedings*, pages 143-149. International Computer Music Association, San Francisco, 1991.

[Desain 91b] Desain, P. and H. Honing. *Towards a calculus for expressive timing in music*. Technical Report, Center for Knowledge Technology, Utrecht, 1991.

[Desain 92] Desain, P. and H. Honing. Time Functions Function Best as Functions of Multiple Times. *Computer Music Journal* 16(2):17-34, Summer, 1992.

[Fey 89] Fey, T. and M. J. Grey. *Using Bars and Pipes* Decatur, Georgia, 1989.

[Grey 75] Grey, J. M. *An Exploration of Musical Timbre*. PhD thesis, Department of Psychology, Stanford University, 1975. Department of Music Report STAN-M-2.

[Hamel 88] Hamel, K. *MusScribe Reference Manual* SoftCore Music Systems, Richmond, BC, Canada, 1988.

[Hild 92] Hild, Hermann and Feulner, Johannes and Menzel, Wolfgang. HARMONET: A Neural Net for Harmonizing Chorales in the Style of J.S.Bach. In Moody, J.E. and Hanson, S.J. and Lippmann,R.P. (editor), *Advances in Neural Network Information Processing Systems (NIPS-4-)*. Morgan Kaufmann, 1992.

[Honing 92] Honing, H. Issues in the Representation of Time and Structure in Music. *Music, Mind, and Machine: Studies in Computer Music, Music Cognition, and Artificial Intelligence.* Thesis Publishers, Amsterdam, 1992, pages 25-40. Also in Proceedings of the 1990 Music and the Cognitive Sciences Conference, Harwood Academic Publishers GmbH, 1992.

[Horowitz 76] Horowitz, E. and S. Sahni. *Fundamentals of Data Structures in Pascal.* Computer Science Press, 1976.

[Jaffe 85] Jaffe, David. Ensemble Timing in Computer Music. *Computer Music Journal* 9(4):38-48, 1985.

[Jordan 86] Jordan, M. I. *Serial Order: A Parallel Distributed Processing Approach*. Technical Report 8604, Institute for Cognitive Science, UCSD, 1986.

[Katayose 89] Katayose, H., H. Kato, M. Imai, and S. Inokuchi. An Approach to an Artificial Music Expert. In T. Wells and D. Butler (editor), *Proceedings of the 1989 International Computer Music Conference,* pages 139-146. International Computer Music Association, San Francisco, 1989.

[Krumhansl 91] Krumhansl, C. L. Music Psychology: Tonal Structures in Perception and Memory. *Annual Review of Psychology* :277-303, 1991.

[Lischka 87] Lischka, C. Connectionist Models of Musical Thinking. In J. Beauchamp (editor), *Proceedings of the 1987 International Computer Music Conference,* pages 190-196. International Computer Music Association, 1987.

[Martens 85] Martens, W. L. PALETTE: An Environment for developing an individualized set of psychophysically scaled timbres. In B. Truax (editor), *Proceedings of the 1985 International Computer Music Conference,* pages 355-365. International Computer Music Association, 1985.

[Mathews 69] Mathews, M. V. *The Technology of Computer Music.* MIT Press, Boston, 1969.

[Mathews 70] Mathews, M. V. and F. Richard Moore. A Program to Compose, Store, and Edit Functions of Time. *Communications of the ACM* 13(12):715-721, December, 1970.

[Maxwell 84] Maxwell, J. T., and S. M. Ornstein. Mockingbird: A Composer’s Amanuensis. *Byte* 9(1):384-401, 1984.

[Nelson 77] Nelson, G. MPL: A Program Library for Musical Data Processing. *Creative Computing* :76-81, Mar.-Apr., 1977.

[Newcomb 91] Newcomb, S. R. Standard Music Description Language Complies With Hypermedia Standard. *Computer* 24(7):76-80, July, 1991.

[Oppenheim 87] Oppenheim, D. V. The P-G-G Environment For Music Composition: A Proposal. In J. Beauchamp (editor), *Proceedings of the 1987 International Computer Music Conference,* pages 40-48. International Computer Music Association, San Francisco, 1987.

[Orlarey 86] Orlarey, Y. MLOGO: A MIDI Composing Environment. In P. Berg (editor), *Proceedings of the International Computer Music Conference 1986*, pages 211-213. International Computer Music Association, 1986.

[Palmer 89] Palmer, C. Mapping Musical Thought to Musical Performance. *Journal of Experimental Psychology* 15(12), Dec, 1989.

[Polansky 90] Polansky, L., P. Burk, and D. Rosenboom. HMSL (Hierarchical Music Specification Language): A Theoretical Overview. *Perspectives of New Music* 28(2):136-179, summer, 1990.

[Pope 91] Pope, S. T. Introduction to MODE: The Musical Object Development Environment. *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology.* In S. T. Pope, MIT Press, Boston, 1991, pages 83-106.

[Puckette 91] Puckette, M. Combining Event and Signal Processing in the MAX Graphical Programming Environment. *Computer Music Journal* 15(3):68-77, Fall, 1991.

[Rodet 84] Rodet, X. and P. Cointe. FORMES: Composition and Scheduling of Processes. *Computer Music Journal* 8(3):32-50, Fall, 1984.

[Rogers 80] Rogers, J., J. Rockstroh, and P. Batstone. Music-Time and Clock-Time Similarities Under Tempo Changes. In *Proceedings of the 1980 International Computer Music Conference*, pages 404-442. International Computer Music Association, 1980.

[Rubenstein 87] Rubenstein, W. B. *Data Management of Musical Information*. PhD thesis, U. C. Berkeley, June, 1987. Memorandom No. UCB/ERL M87/69.

[Scaletti 89] Scaletti, C. The Kyma/Platypus Computer Music Workstation. *Computer Music Journal* 13(2):23-38, Summer, 1989.

[Scaletti 91] Scaletti, C., and K. Hebel. An Object-based Representation for Digital Audio Signals. *Representations of Musical Signals.* In G. De Poli, A. Piccialli, and C. Roads, MIT Press, Cambridge, Mass., 1991, pages 371-389, Chapter 11.

[Scaletti 92] Scaletti, C. Polymorphic transformations in Kyma. In *Proceedings of the 1992 ICMC*, pages 249-252. International Computer Music Association, San Francisco, 1992.

[Scarborough 89] Scarborough, D., B. O. Miller, and J. A. Jones. Connectionist Models for Tonal Analysis. *Computer Music Journal* 13(3):49-55, Fall, 1989.

[Schottstaedt 83] Schottstaedt, Bill. Pla: A Composer's Idea of a Language. *Computer Music Journal* 7(1):11-20, Spring, 1983.

[Sejnowski 86] Sejnowski, T. J. and C. R. Rosenberg. *NETtalk: AParallel Network that Learns to Read Aloud*. Technical Report JHU/EECS-86/01, Johns Hopkins University, 1986.

[Shepard 82] Shepard, R. N. Geometrical approximations to the structure of musical pitch. *Psychological Review* 89(4):305-333, 1982.

[Simon 68] Simon, H. A. and Sumner, R. K. Pattern in Music. *Formal Representation of Human Judgment.* In B. Kleinmuntz, Wiley, New York, 1968.

[Taube 89] Taube, H. Common Music: A Compositional Language in Common Lisp and CLOS. In T. Wells and D. Butler (editor), *Proceedings of the 1989 International Computer Music Conference*, pages 316-319. International Computer Music Association, 1989.

[Teitelbaum 85] Teitelbaum, R. The Digital Piano and the Patch Control Language System. In W. Buxton (editor), *Proceedings of the International Computer Music Conference 1984*, pages 213-216. International Computer Music Association, 1985.

[Todd 91] Todd, P. M. and D. G. Loy. *Music and Connectionism.* MIT Press, Boston, 1991.

[Waters 90] Waters, S. and T. Ungvary. The Sonogram: A Tool for Visual Documentation of Musical Structure. In S. Arnold and G. Hair (editor), *ICMC Glasgow 1990 Proceedings*, pages 159-162. International Computer Music Association, 1990.

[Wessel 85] Wessel, D. L. Timbre Space as a Musical Control Structure. *Foundations of Computer Music.* In C. Roads and J. Strawn, MIT Press, Cambridge, MA, 1985, pages 640-657. Originally published in *Computer Music Journal* 3(2):45-52, 1979.