Geraint Wiggins,*† Eduardo Miranda,*† Alan Smaill,* and Mitch Harris*
*Department of Artificial Intelligence
†Department of Music
University of Edinburgh
Edinburgh, Scotland, United Kingdom
{geraint, E.Miranda, A.Smaill,
M.Harris}@Edinburgh.ac.uk

A Framework for the Evaluation of Music Representation Systems

In this article, we provide a framework for the description and evaluation of music representation systems suitable for implementation on computers. Our main concern is with representational aspects, rather than with implementation; however, if a system is to be useful, a good implementation is required.

A representation system may be suited to many different purposes, and its usefulness is relative to the task at hand. It is not possible to cover all such purposes, but we can distinguish three general sorts of tasks: recording, analysis, and generation/composition. In recording tasks, the user wants a record of some musical object, to be retrieved at a later date. Accuracy is the prime concern. The analytical user wants to retrieve not the "raw" musical object, but some analyzed version of it, revealing some salient feature. The ability to find or exploit structure within the object is important. For tasks in music generation and composition, the user wants to build a new musical object, either from scratch or by transformation of an existing object. Manipulability and flexibility of the representation are needed. The classification we present below is oriented toward these three situations.

The structure of the article is as follows. After an outline of the concepts we propose for consideration and of the background ideas involved, we will discuss the possibility of using a general-purpose representation language developed in the field of artificial intelligence (AI), not specifically targeted at musical applications. We then consider a representative selection of music representation systems with respect to the concepts we consider relevant. We restrict our attention to systems that work on the level of notes and more abstract structures, rather than systems for dealing with, for example, description of timbre. Some systems raise further issues. In particular, "connectionist" systems use a different notion of representation from most of those we describe and do not fit so easily into our framework. This is discussed after the main survey. The article concludes by summarizing why we consider this analysis of music representation systems to be useful.

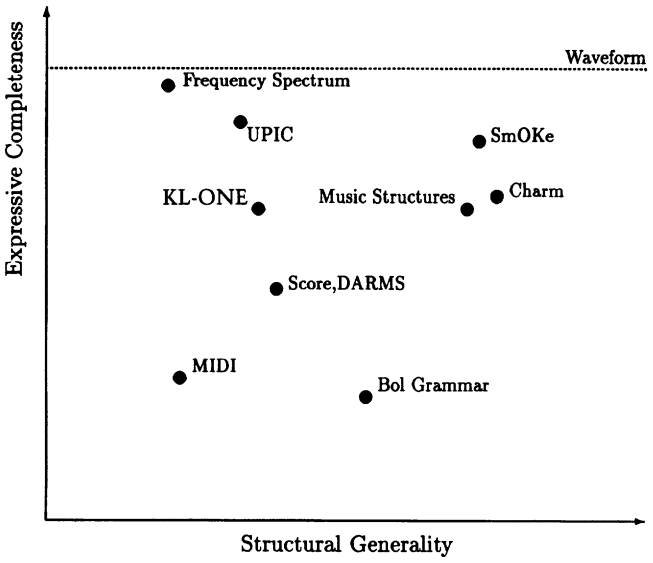
Computer Music Journal, 17:3, pp. 31–42, Fall 1993,

Two Dimensions of Representation Systems

We consider the relative merits of different systems along two orthogonal dimensions: *expressive completeness* and *structural generality*. Expressive completeness refers to the range of raw musical data that can be represented, and structural generality refers to the range of high-level structures that can be represented and manipulated. For example, at one extreme we can use a waveform to represent any (particular) performance at the cost of being unable to capture abstract musical content easily; a very similar performance of the same piece may look very different. For another example, MIDI encodings capture some generality about a performance (at the cost of losing some completeness) but do not extend to the expression of general high-level structures. Traditional score-based notation has more structural generality than MIDI, giving some tonal and metric information, but is restricted in expressive completeness to traditional Western tonal music.

Figure 1 shows several well-known representation systems classified according to their position with respect to these two dimensions. We believe it is useful to evaluate the suitability of a representation system for a particular task by relating the task and the system to these two dimensions. We will discuss each of the systems shown in Fig. 1 below.

Fig. 1. Two dimensions for comparison of music representation systems. The axes and systems are described in the text.



What Is Represented?

First, we must be clear about what we are trying to represent. We draw a distinction between a *score* (in the conventional sense) and a *musical object*. A score may be thought of as instructions to a musician, computer system, or whatever, to be read as the basis for the realization of a piece of music, while the result obtained from this process, and its subparts, are musical objects. To put this another way, a score (usually) only partially defines the musical object produced when the scored piece is performed; the score "precedes" the realization or interpretation (Nattiez 1975, pp. 109–117).

Even though our evaluation strategy may be applied both to scoring systems and to representations of pure musical objects, confusion will arise if we mix the two. Therefore, for this discussion, we will focus on representations of musical objects, which gives us a broader spectrum to consider.

A further potential confusion is that any representation of a musical object can be viewed as a score and made open to "reinterpretation"; conversely, scores can be (and often are) viewed as representations of musical objects. This distinction should be borne in mind when reading the discussion below.

Procedural and Declarative Representations; Objects

There is a distinction, which will be useful in the forthcoming discussion, between programming languages and between data representations that are either *procedural* or *declarative*. We characterize the difference for programming languages as follows. Procedural languages (e.g., Fortran or C) require us to state how something is to be done. On the other hand, declarative languages (e.g., Lisp or Prolog) allow us to specify what is to be done or what is true—execution is left to the programming environment and is (in theory) not the concern of the programmer.

In more concrete terms, the procedural programmer uses a system that follows basic instructions about the manipulation of data, for example, "add 1 to 2" or "put it in X." The declarative programmer, on the other hand, works in a rather different way. In functional languages, like Lisp, one thinks in terms of a result, obtained by the evaluation of a function, so instead of saying "add 1 to 2," one would say "the sum of 1 and 2." The connection between the concept of a sum and the action of adding is made by the evaluation mechanism. In logic programming languages such as Prolog, a programmer specifies logical relations between data, so that "the sum of 1 and 2 is 3" is a simple program. Logical inference is used to find solutions to queries (for example, "what is the sum of 1 and 2?" or "what number added to 1 gives 3?") posed to the programming system. This is a very high-level specification—stating what is true, rather than giving explicit instructions to manipulate data.

Now consider the distinction for knowledge representations. If we have a program that will generate a declarative representation of a musical object, we can say that the program itself is a procedural representation of that object.

Why then should we not use the program itself, instead of the output it yields, for our purposes of evaluation? We do this because, practically speaking, it can be very difficult to get at the data implicit in a program, especially a procedural program, without running it. Anyway, if we run the program, we end up using the explicit data it generates.

The issue of procedural-versus-declarative representations will arise later, but we propose to defer it,

since the majority of representations are declarative, and we wish to compare like with like. There is no point in discussing generation programs when we will necessarily discuss the form of their output.

We must also briefly discuss the nature of object-oriented programs and representations. It is often thought that the object-oriented programming paradigm is an alternative to the declarative and procedural ones, or to the procedural, functional, or logical basis of program design. This is not the case. Object orientation is a style of programming, which may be implemented in any of the above forms, to varying degrees of advantage. Specialized object-oriented languages may be procedural, functional, or logical, but they have built in to them certain structures and operations that facilitate the object-oriented approach to program design.

An object-oriented program is stated in terms of "objects," which are localized collections of data and procedures. Objects may be arranged in a hierarchy, and various forms of property inheritance may be defined. Objects have their own local variables and memory, and may share a view onto common memory with other objects. The behavior of the objects in a program is determined by messages sent between them, defined by the programmer. We will discuss object orientation further in the section on the SmOKe system below.

General-Purpose Representation Systems

Before looking at special-purpose music representation systems, we consider general-purpose systems for representing and manipulating knowledge. What do general-purpose systems provide, and are musical applications likely to raise problems that demand special treatment?

A good knowledge representation language lets the user express knowledge of a given domain naturally and concisely and supports an efficient reasoning regimen to retrieve and manipulate the knowledge encoded. Much work has gone into the development of such formalisms, which allow a declarative reading of the encoded knowledge (Brachman and Levesque 1985).

KL-ONE (Brachman and Schmolze 1985) has been used as the basis for a musical application (Camurri et al. 1992). It allows the user to describe a domain in terms of "concepts." These describe both the individual objects in the domain and classes or "sorts" of objects. Objects can belong to several sorts because some sorts are more general than others (so if Freddie is of sort "shark," he is also of sort "fish," and more generally again "animal"). This means that general knowledge can be attached to all objects of a given sort ("all animals eat") and used whenever an object is of the appropriate sort (so, "Freddie eats"). This sort of inference is called "inheritance" and is important because it allows sharing of knowledge within taxonomies of sorts, and can also be implemented efficiently. Concepts are defined in terms of their internal structure—KL-ONE uses "roles" for the constituent parts or attributes of a concept, and "structural descriptions" that express in logical terms how the roles interrelate for a particular concept. Brachman ensures that KL-ONE is a declarative representation, and it is important that the representation can be efficiently manipulated, with an appropriate interpreter.

Other general-purpose representation systems exist. So can we simply use a general-purpose system to represent musical objects, or are there special characteristics involved that raise special problems?

Some musical systems treat the notion of time in a special manner when describing musical objects (Balaban 1988; Diener 1989). This can be for pragmatic reasons (efficient access to the temporal properties of objects), or because time is held to be the fundamental axis in musical descriptions. However, we can also regard time simply as one dimension among others in our representation. Pieces like Olivier Messiaen's *Modes d'intensités et valeurs* illustrate how time and pitch dimensions (among others) can play equal organizing roles. So the temporal aspects of music do not require special representation techniques (though it may help to optimize the treatment of time).

One feature of music representation, whether for analysis or generation, that is not common outside music is the importance of multiple viewpoints—the ability to represent the same musical object in many

different ways for different purposes. In many representation tasks, ambiguity of representation can be a problem, whereas in music, multiple readings of an object can be vital. A system in which this was difficult would rate poorly in terms of structural generality. Although general-purpose systems can be used to express such multiple viewpoints, the importance of this aspect is such that it should be explicitly supported, as it is in some of the systems we examine below. Apart from this, however, the problems raised in music representation are not so different from the general case.

Evaluating Music Representation Systems

This section is a survey of specialized music representation systems, evaluated in terms of our two proposed dimensions. There are many more systems than we can possibly cover here, so we have tried to choose a representative member for each category. No slight is intended on those systems not mentioned.

Each system will be outlined and assessed in terms of expressive completeness and structural generality. The point of this exercise is twofold. First, it enables us to exemplify what we feel is important about musical knowledge representation in general and about the individual systems in particular; second, it allows us to explain our evaluation strategy and its parameters.

Spectrum Analysis

We have already mentioned the raw sound waveform as an example of maximal expressive completeness and minimal structural generality. We can carry this abstract discussion further to explain our axes of evaluation. Consider the output of a spectrum analyzer applied to the sound waveform. The result of the analysis is a three-dimensional mathematical structure, which can be displayed as frequency versus time versus amplitude. The difference between this representation and the raw waveform is that the individual partials are separated out from the inscrutable (and so structurally nongeneral) original wave. It is now much easier to isolate, for example, individual notes (by searching for groups of partials in harmonic frequency ratios) than it was in the raw wave. We can draw notional circles around harmonics that form meaningful groups, which we could not do before. Note, however, that this notional grouping is outside the representation—the spectrum—itself, and so does not contribute to its structural generality. The point is that, in the spectrum form, the information in the wave is much more readily available for analysis and editing than it was before, in a form that is musically more meaningful.

Further, if we have a perfect spectrum analyzer and a perfect additive synthesizer to reconstitute our waveform after analysis, we can exactly reproduce the original wave from the spectrum representation. Thus, even though we have increased structural generality, we have lost no expressive completeness—our two axes of evaluation are mutually independent.

MIDI

The MIDI system (Rothstein 1992) is a combined hardware and software communications protocol for electronic music systems. Recent developments of the idea include definitions for "MIDI files" that contain all the data required to play a "song" on a particular synthesizer or other MIDI instruments. Some instruments have "system exclusive" commands to allow the dumping of specific information about their settings to other instruments or to archive storage. In the context for which it was originally intended, the MIDI system functions well. However, as a music knowledge representation, it scores few points in either structural generality or expressive completeness.

A MIDI "event" is the receipt or transmission of a number of bytes of information by an instrument or computer. Two common events are Note On and Note Off. Each consists of three parts: an instruction (Note On or Note Off), a note number (pitch, based on a standard numbering of the piano keyboard), and a velocity (loudness, the speed with which the note

is depressed or released). Time is implicit; a note begins on receipt of a Note On event, and ends on receipt of a Note Off.

Let us first consider expressive completeness. MIDI time is represented implicitly in terms of "real" time—or, at least, the ticks of a notional clock. This means that it is as close to what was played as the granularity of the clock ticks will allow and, therefore, potentially quite high in expressive completeness. On the other hand, a huge amount of pitch information is abstracted out of the representation. This is due to the approximation of pitches to piano keys. No explicit assumption is made about tuning systems; equal or just temperament use the same note representation in a MIDI file, but the representation does not encompass both; it acknowledges neither, simply ignoring the issue. On balance, then, in terms of expressive completeness, MIDI sits a long way below the spectrum representation discussed above.

As for structural generality, even the very latest versions of the MIDI system score badly. While the MIDI concept of "note" and its start time and end time are certainly more structurally general than the spectrum representation, there is no further allowance for structural annotation until one reaches the level of "MIDI file," which is intended to encapsulate a whole "song." The pitch dimension is equally homogeneous. Thus, the kind of multilevel and multiview representations discussed below are impossible.

Score Representations: DARMS

For completeness, we must cover a computer scoring system, which, like any scoring system, can be used or abused as a representation for musical objects. One such is DARMS (Erickson 1977). DARMS is a language that allows representation of traditional Western scores in the way considered most computer-friendly at the time of its inception—that is, as ASCII letter codes.

The DARMS representation scheme encourages the user to view scores as an uninterpreted collection of graphic symbols—so that a curved line, for example, denotes neither a phrase nor a slur, but is simply a graphic symbol placed somewhere on a piece of paper. For this reason, DARMS and any other system like it, for representing traditional scores in computer-friendly form, occupy exactly the same place in Fig. 1 as the score itself.

Graphical Representations: The UPIC

There is no difference in potential expressive power between a graphical representation and a textual one. That is to say, any information represented graphically can necessarily be recast in a more traditional, nongraphical form. Why, then, use a graphical representation? Simply because the information represented can be substantially more easily available to the human eye. Given this, we can straightforwardly apply our evaluation criteria to graphical systems exactly as to nongraphical ones.

A good example of graphical representation systems is that used in the UPIC, a computer musical instrument intended to embody some of the ideas presented by Xenakis (1992). The UPIC is a synthesizer incorporating a large graphics tablet on which lines can be drawn. The drawings can represent waveforms, amplitude envelopes, and events (i.e., notes), expressed as signal, amplitude, or pitch, respectively, graphed against time. Each UPIC event is a notionally continuous pitch gradient with fixed start points and end points and is associated with a waveform and envelope. Pages denote collections of events; scores are sequences of pages.

The UPIC representation is a hierarchy—the first we have seen here. The graphical view can be applied at several levels: waveform, envelope, and score. However, at the level of our interest here, where the primary events are (roughly speaking) notes, the system is hardly hierarchical at all. Each page of score is stored separately from its neighbors and may be manipulated, just as one would expect to manipulate images with a very simple WYSIWYG (what you see is what you get) graphics program. Beyond this, there is no concept of structure other than that implicit on the pages of the score. So while the UPIC scores well on expressive completeness, because of the high

granularity and flexibility of its basic concept of "note," it is only slightly more structurally general than MIDI.

Music Programming Languages

Programming languages, specialized or otherwise, are increasingly being used for tasks related to music analysis, generation, and composition. Specialized languages often consist of libraries of functions, extending a general-purpose programming language, and a customized environment for user interaction. It is therefore important to consider carefully the contribution of the host language when one evaluates such systems.

Recall that we have distinguished two kinds of programming languages: procedural (e.g., Fortran, C); and declarative (e.g., Lisp, Prolog). Recall also that we restrict our attention to systems that treat music on the level of notes, and no lower, so we will not discuss the sound-generation mechanisms of the languages covered below, but cover only representation of note events.

We have not placed the music programming languages per se in Fig. 1, because they are different in kind from the representation systems shown there. Nevertheless, it is possible to discuss the representations they use.

The Music-N Family

The Music-N family began at AT&T Bell Laboratories (Matthews 1969; Pope 1993). It has spawned descendants such as Music V, Music-11, Csound, cmusic, and, recently, CLM (Schottstaedt 1992).

These languages use a two-part music representation, consisting of an *orchestra* of sounds to be used and a *score* of notes to be played. The two are stored separately, so one orchestra may perform many scores. Both parts of the music representation are declarative—the orchestra part specifies connections inside a sound generator built from a number of predefined blocks, and the score part is a list of note specifications, each with start time, duration, pitch, and other parameters. Running the "program" is equivalent to "performing" the score, the output being a digitized representation of the sound waveform. Note that the only "interpretation" involved is in the orchestra; the resulting sounds follow the specification literally, so the combined specification might be said not to be a score in the traditional sense; rather, it defines a musical object directly.

In expressive completeness, the Music-N family fares quite well, since parameters are expressible with fine granularity and therefore can be made to produce or reproduce a musical object very accurately. However, structural generality is less satisfactory; notes may be grouped in "sections," but one cannot specify relationships between sections, nor parameterize the sections themselves. No hierarchical arrangement is possible beyond the section level.

Common Music

Common Music (Taube 1991) is a language that allows one to write programs that generate sequences of notes in a variety of formats—such as MIDI events or Csound notes. It is a descendant of PLA and SCORE (Loy and Abbott 1985) and is implemented as an extension of Common Lisp, providing a set of tools that perform operations on lists of musical parameters. Note that here is our first example of a "procedural" representation of musical objects. Even though Lisp is a declarative programming language, the specification of the music arises from the evaluation (i.e., the execution) of the program—there is not in general a declarative representation of notes but, rather, of the processes that generate them.

Assessment of Common Music with respect to our two parameters is complicated by the fact that, regardless of whatever representation it has of its own, on evaluation it gives rise to a representation of the produced musical object in some other (programmer-selected) system. The internal representation of Common Music is in terms of standard Lisp data types, so it is in principle as strong or as weak as any of the systems here described, depending on the programming style used. However, because the final output must be restricted to the terms of another system, the actual value is that of the chosen output representation. The same applies to structural gener-

ality; while the generality of the selected output system limits the overall generality of a given Common Music program, the original data produced can be highly structurally general, with arbitrarily complicated annotations and grouping being created by the Lisp program.

Stella

Stella (Taube 1992) uses a frame-like representation (Minsky 1981), enhancing Common Music to admit both implicit procedural and explicit declarative specification of musical objects. Musical knowledge is built around *Stella objects*. A Stella object is either an atomic element that reflects a basic compositional datum (e.g., a single note) or a more abstract concept denoting a collection of elements or of other collections (e.g., a monophonic sequence of notes), or a frame. A frame is a data structure with components called slots. Slots have names and accommodate information of various kinds, such as elements, collections of elements, references to other frames, or procedures to compute the slot values. Various theories of inheritance and default may be applied to data represented in frames.

The frames make little difference to the evaluation of Stella's output, compared with that of Common Music, because one is still restricted by one's chosen output representation. However, the structural generality of the internal representation is much improved by the addition. It is now possible explicitly to annotate groupings and relationships and, importantly, to use that information as part of the musical object construction. As well as (and because of) the increased structural generality, user transparency is significantly improved over the implicit procedural representation of Common Music.

Summary

In this section we have shown that, when evaluating a (music) programming language we must draw a distinction between the language itself and its output. Also, declarative programs do not always represent knowledge declaratively, and although declarative programs do not always lead to structurally general representations, a declarative knowledge representation (even in a fully procedural program) can increase structural generality significantly.

Grammar-Based Approaches: The Bol Processor

A grammar is a means of describing the structure of a class of syntactic entities. Grammars may be implemented in many ways, the basic idea being that structure of larger entities is described in terms of their subparts. For example, in English, a sentence can be composed of a noun phrase followed by a verb phrase. This is often written

$$\text{Sentence} \rightarrow \text{Noun-Phrase Verb-Phrase},$$

to form, along with definitions for other syntactic structures, a "phrase structure grammar." Other styles of grammar exist, such as *categorical grammars*, in which each word is in a syntactic "category," some of which are functions. Rules are used to define how members of categories may be combined to produce members of other categories. For example, a noun phrase might be in category NP, and a verb phase in $S\backslash NP$; then, given the rule

$$X + Y \backslash X \rightarrow Y$$

the two combine to form a sentence, S. Computational linguists often use syntactic analysis by grammar (parsing) to determine the structure of a sentence, the words of which are then translated and recombined to give a representation of the meaning (semantics) of the sentence in a machine-friendly form (e.g., predicate logic). A grammar for a class of structures may be used to generate those structures, to check if a given structure falls within the class described, or just for the description alone; the structure itself is purely declarative. It is often possible to use a given grammar for any or all of these purposes. Use of a grammar may be more or less computationally difficult, according to its expressive power (Winograd 1972).

The use of linguistic tools for music begs a deep philosophical question. If there is an analogy between the syntax of language and musical structure, what, if any, is the relationship between linguistic

semantics and the "meaning of music"? Indeed, it is by no means clear that such "meaning" exists. However, this issue is outside the scope of the current article.

Musical grammars are not usually intended to represent individual compositions, but to describe classes of compositions (e.g., in a particular style or form). As such, they are not the same as the other systems covered here, which are mostly intended for representing the kind of information that results from using a grammar. Therefore, grammars are more appropriate for generating and analyzing music than for recording.

Although the grammar-based approaches do not exactly correspond with the rest of our examples, our two dimensions can help in evaluating them, and many of the same issues arise as with the other examples. The "Bol Processor," BP1 (Bel and Kippen 1992), is a good example of a grammar-based system. It generates pieces of music (*qa'idas*) for Northern Indian tabla drumming; according to expert evaluators, its output is credible. BP1 works from a top-down, phrase-structural analysis of the *qa'ida* form, subdividing down to the level of individual note sequences. A problem for any characterization of musical forms involving repetition is representation within the grammar of the connections between repeated parts. To achieve this, Bel and Kippen define a form of grammar called a pattern grammar, which has rather more than the descriptive power of the context-free grammars used to define most programming languages.

BP1 grammars fall low on the scale of expressive completeness; they only represent the tones of the tabla drums and not a full pitch metric. This, however, is a limitation that has been designed; it is not, in context, a drawback. Structural generality, however, is high, because of the grammar's ability to express encapsulation of structures into higher level structures. BP1 is particularly structurally general, by comparison with similar systems, because it can explicitly represent connections between different sections of music.

Roads (1979) presents a good survey of grammar-based representations. Lerdahl and Jackendoff (1983) give a detailed discussion of a particular system.

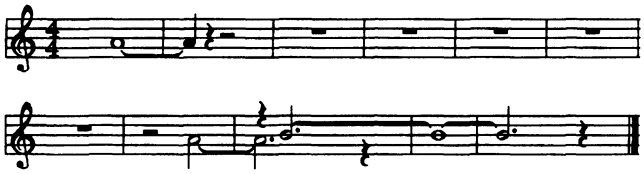
Music Calculi: Balaban's "Music Structures"

A good example of the notion of a "music calculus" is Mira Balaban's "Music Structures" (Balaban 1992). The point of designing a calculus for knowledge representation is that one would like a language for that representation that allows general expression, but also admits unambiguous inference about the information represented. Balaban's central idea is that music must be represented in terms of the interleaving of its temporal and hierarchical properties. She approaches the representation task by means of hierarchies of structures, abstracted along a set of parallel time lines, and interconnected by various "concatenation operators," all defined in terms of one basic operator, "musical concatenation," •, which is essentially the set insertion operator.

This yields a powerful and flexible representation system, which is too complicated to explain in detail here. Instead, we give an example, borrowed from Laske , Balaban, and Ebcioglu (1992) . Consider the following music structure, presented in a simplified form, which treats • like the "cons" function in Lisp, so (*ms1* • (*ms2* • NIL)) is written as (*ms1 ms2*). The musical structure

([a,5]@–20 ([b,10]@5 [a,5]@2)@8)

represents the following sequence of notes (rests are implicit; we have added the time signature and bar lines for legibility).



The lowest-level music structures are of the form [*ms,d*], where *ms* is some music structure (here, a note of the Western scale) and *d* is its duration expressed as a real number. The @ ("time-stamp") operator links a music structure with a real start time; times within the structure are then relative to that start point. Finally, the • operator associates two music structures together in the temporal relation defined by their start times. So the example above

denotes a compound music structure made up of an atomic event (a 5-beat "a") and another structure, which consists in turn of a 10-beat "b" and an overlapping 5-beat "a." An example of an operation one might perform on this is flattening—making it into a structure only one level deep, as in,

([a,5]@–20 [b,10]@13 [a,5]@10).

The resulting structure is now ready for conversion into MIDI signals, for example.

We can assign names to music structures, and, because the time-stamp operator is relative, we can use the same structure many times (in a motivic piece of music, for example). The user is limited only by imagination in terms of hierarchies used in the representation and of their attached significance. Such significance can be made explicit, by means of *attributed music structures*, arbitrary labelings attached to music structures.

One advantage of Balaban's system is its open-endedness. The symbols used are mostly user-chosen and so are open to free interpretation, though this can lead to the construction of ad hoc operators, such as Balaban's "~" (overlapping horizontal concatenation) operator. This extensibility means that the system can have a high expressive completeness, qualified by the comment that it is in a sense by default, because there is no explicit extensibility. Balaban does not address expressive completeness in her examples, though she does point out that they use the "twelve tones" system, which perhaps implies that she expects to use different symbols to express other tonal systems.

In terms of its structural generality, Balaban's system fares equally well. Her hierarchies are designed specifically to maintain that property.

A significant drawback with the Music Structures system, related to the issue (above) of the choice of symbols to represent (for example) different tuning systems, is the use of the real number line as the representation of time. Although we believe Balaban is correct to state that the required mathematical properties of the real numbers give a possible characterization of time, musicians rarely think in those terms. An improvement would be to use an algebra with the relevant properties of the real numbers, but with abstract syntax. This could add to both the system's structural generality and its expressive completeness. This is the approach taken in the Charm system, described below.

Object-Oriented Music Representation: SmOKe

We outlined the basic notions of the object-oriented programming style above. In this section, we discuss an object-oriented music representation system that is independent of implementation language and describe how it relates to the other systems covered here and to our evaluation parameters.

The SmOKe (Smallmusic Object Kernel) system of Pope (1992a, 1992b) is a universal scheme for music representation. By this we mean that SmOKe is not in itself a representation for music, but a specification for what a music representation should be. This specification is object-oriented, meaning it is presented in terms of class hierarchies of objects. Objects "share state and behavior and implement the description language as their protocol." One actual implementation of SmOKe is explained by Pope (1992b) using the uniform object-oriented Smalltalk-80 programming language.

Many of the wide-ranging and powerful capabilities of SmOKe are outside the scope of our example evaluation for this article. For example, SmOKe requires representation of timbre in a number of standard forms, and descriptions of "instruments" that map data in a SmOKe representation into control signals for synthesizers or music programming languages. It also admits scores, including traditional Western score notation.

Implementations of SmOKe fare well in expressive completeness (at the note level). Descriptions of note events are given in terms of abstract properties—see the section on Charm, below, for a discussion of this idea—though it is not specified how this is to be implemented. Since an abstract specification can describe parameters to an arbitrary level of detail, the user decides how expressively complete an implementation of SmOKe must be.

To consider SmOKe's structural generality, we need to know what hierarchical structures are avail-

able. We emphasize that it is not the object-oriented nature of the description that makes the representation structurally general—the objects are only a means of writing the information down. Frames, for example, would in principle do just as well. SmOKe's hierarchy gives us arbitrary nesting of structures—groups of events may be specified and mixed with events and other groups to form higher level groups. It also provides "abstractions for the descriptions of 'middle-level' musical structures (e.g., chords, clusters, or trills)." This, while apparently adding to structural generality, may in fact be restricting it, since it is possible to specify these kinds of relationships in general logic terms. The existence of particular instances of groupings may suggest that general specification of such things is not possible. We suggest, then, that SmOKe is highly structurally general, but that annotation of its structures may not be as general as that in, for example, Charm (see below).

Abstract Representation: Charm

The Charm (Common Hierarchical Abstract Representation for Music) system (Harris, Smaill, and Wiggins 1991) is an attempt to free the representation of music (subject to some experimental constraints) from application- or domain-specific influence. It is intended to allow representation of music in any terms desired by the user. This is made possible by separation of the "concrete" representation actually used by a musician or a program from the "abstract" mathematical properties required of it. The technique will be familiar to computer scientists—the end result is an abstract data type (ADT). Charm defines the notion of musical events, abstract data types for the properties of events, and a system for building hierarchies of events to describe music.

Charm events are notes of constant pitch or frequency, with a start time and duration, intensity, and place holders for other information (e.g., timbre) not currently provided for. The constant pitch requirement is an approximation to reality to allow for tractable experimentation while still admitting a substantial corpus of real examples. Both pitch and time are described in terms of data structures that are arbitrary except that they must obey certain mathematical rules—those of a linearly ordered commutative group (Harris, Smaill, and Wiggins 1991). Charm defines names for the operations on the representation that must be supplied—for example, `Pitch` and `Duration` are functions that, given the name of an event, return values of its property; it is left to the designer or user of a given concrete representation to build the necessary implementation. Any program using Charm will be able to access the user's representation via the defined functions (though, of course, one is always restricted by the aptness of one's data for one's program. For example, applying a Bach-style harmonization program to a raga is unlikely to produce useful results, regardless of Charm's interfacing capabilities).

The point is that the ADT approach allows us to represent as much detail about (constant) pitch as we like, in whatever form we like, so long as we follow the mathematical rules. For example, an analysis program designed for the twelve-tone scale was used on quarter-tone music without changing the code of the analyzer(Smaill and Wiggins forthcoming). This was possible because both program and representations were built using Charm. On this basis, Charm rates very highly in terms of expressive completeness because, subject to the temporary experimental constraints placed by the designers, we can represent whatever we want—the abstraction approach allows us to choose exactly the mathematical properties we need.

Charm events can be grouped by the construction of *constituents*. These are arbitrary collections of events or other constituents, known as *particles*, and are referred to via unique labels generated by any given implementation of Charm. Each constituent also has a unique name and may be labeled with a set of first-order logical formulae describing the properties of its particles or of the constituent as a whole. The definitions may override defaults, which are globally defined, such as the start time and duration of a constituent. It is also possible to label a constituent "definitionally"—to state that the constituent itself defines something (e.g., a piece or motive). Finally, the user may attach an arbitrary text string to each constituent to express any other information. No inference from this information is assumed possible.

Because of the use of arbitrary logical formulae in the constituent specification, the structural generality of Charm is very high, as any property of or relationship between constituents can be represented explicitly.

Symbolic and Subsymbolic Representation

Some approaches to information processing and representation differ qualitatively from the discrete symbolic manipulation that characterizes most of the systems covered here. One important class is that using parallel distributed processing (PDP, or connectionism) (Leman 1988; Todd and Loy 1991).

Although PDP systems were originally inspired by quasi-biological models of neural networks, modern connectionism is heavily rooted in statistics and the study of dynamic systems. They are particularly useful for implementing processes of categorization and matching with low-level data.

Symbolic and PDP approaches are not mutually exclusive; it is likely that all three would be needed in a comprehensive model of music cognition. The key to understanding their mutual relevance lies in the concept of abstraction boundaries. As we explained in the last section, the abstraction boundary we have chosen for the Charm system is the one of note events. Above this level we can construct a meaningful symbolic system; any hypothesized representations or processes below it are subsymbolic with respect to our abstraction boundary. PDP representations might well be appropriate (e.g., for identifying musical events). Furthermore, PDP systems are not intrinsically subsymbolic; this depends where they lie within a system with respect to an abstraction boundary, and in a many-layered system there may well be more than one.

Conclusion

The main focus of this article has been the representational adequacy of different approaches to music representation. As researchers in artificial intelligence and cognitive psychology are painfully aware, representation is a thorny issue. One clear lesson that has been learned, however, is the importance of assessing notation in the context of a whole "representational system" (i.e., not just the notation but also the processes that act on it). Therein lies the problem for the would-be constructor of a general-purpose system of notation—one simply cannot anticipate all the purposes to which it may be put. This problem will be familiar to anyone serving in a standards institution such as ANSI.

Alongside this assessment, we have established some criteria by which the representational adequacy of systems may be judged. Although undoubtedly useful, systems such as DARMS and MIDI are primarily communications protocols, which are not suited for the representation of high-level musical structures. In contrast, we suggest that systems such as Balaban's Music Structures, Pope's SmOKe, and our proposed Charm representation, while having a significant degree of expressive completeness, go beyond being communications protocols; they are first steps toward creating expressive, general music representation languages. By this, we mean systems that allow the expression and manipulation of both established and novel musical structures while maintaining their ability to represent raw musical data.

We have isolated two orthogonal dimensions along which these properties may be measured, giving a means of judging systems (both existing and to-be-designed) as to their suitability for particular purposes. For maximal utility in a given system, one would wish to maximize both dimensions. However, in the design of an efficient communication protocol, such as MIDI, one is more likely to emphasize expressive completeness, rather than structural generality. We suggest, then, that these dimensions may be a useful guide in choosing an existing representation system for a particular purpose and in designing systems for future use.

References

Balaban, M. 1988. "A Music Workstation Based on Multiple Hierarchical Views of Music." In *Proceedings of the 1988 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 56–65.

Balaban, M. 1992. "Music Structures: Interleaving the Temporal and Hierarchical Aspects in Music." In O. Laske, M. Balaban, and K. Ebcioglu, eds. 1992. *Understanding Music with AI—Perspectives on Music Cognition*. Cambridge, Massachusetts: MIT Press, pp. 110–139.

Bel, B., and J. Kippen. 1992. "Bol Processor Grammars." In O. Laske, M. Balaban, and K. Ebcioglu, eds. 1992. *Understanding Music with AI—Perspectives on Music Cognition*. Cambridge, Massachusetts: MIT Press, pp. 366–401.

Brachman, R. J., and H. J. Levesque, eds. 1985. *Readings in Knowledge Representation*. Los Altos, California: Morgan Kaufmann.

Brachman, R.J., and J. G. Schmolze. 1985. "An Overview of the KL-ONE Knowledge Representation System." *Cognitive Science* 9:171–216.

Camurri, A., A. Frixione, C. Innocenti, and R. Zaccaria. 1992. "A Model of Representation and Communication of Music and Multimedia Knowledge." In *Proceedings of the Tenth European Conference on Artificial Intelligence*. Chichester, England: John Wiley and Sons, pp. 164–168.

Diener, G. 1989. "TTrees: Tool for the Compositional Environment."*Computer Music Journal* 13(2):77–85. Also in S. T. Pope, ed. 1991. *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*. Cambridge, Massachusetts: MIT Press, pp. 157–170.

Erickson, R. F. 1975. "The DARMS Project: A Status Report." *Computing and the Humanities* 9(6):291–298.

Harris, M., A. Smaill, and G. Wiggins. 1991. "Representing Music Symbolically." In *Proceedings of the IX Colloquio di Informatica Musicale*. Venice: Associazione di Informatica Musical Italiana, pp. 55–69.

Laske, O., M. Balaban, and K. Ebcioglu, eds. 1992. *Understanding Music with AI—Perspectives on Music Cognition*. Cambridge, Massachusetts: MIT Press.

Leman, M. 1988. "Symbolic and Sub-symbolic Information Processing in Models of Musical Communication and Cognition." *Interface* 18:141–160.

Lerdahl, F., and R. S. Jackendoff. 1983. *A Generative Theory of Tonal Music*. Cambridge, Massachusetts: MIT Press.

Loy, D. G., and C. Abbott. 1985. "Programming Languages for Computer Music Synthesis, Performance and Composition." *ACM Computing Surveys* 17:235–265.

Matthews, M. 1969. *The Technology of Computer Music*. Cambridge, Massachusetts: MIT Press.

Minsky, M. 1981. "A Framework for Representing Knowledge." In J. Haugeland, ed. *Mind Design*. Cambridge, Massachusetts: MIT Press, pp. 95–128.

Nattiez, J.-J. 1975. *Fondements d'une sémiologie de la musique*. Paris: Union Générale d'Éditions.

Pope, S. T. 1992a. "The Interim DynaPiano: An Integrated Computer Tool and Instrument for Composers." *Computer Music Journal* 16(3):73–91.

Pope, S. T. 1992b. "The Smallmusic Object Kernel: A Music Representation, Description Language, and Interchange Format." In *Proceedings of the 1992 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 106–109.

Pope, S. T. 1993. "Machine Tongues XV: Three Packages for Software Sound Synthesis." *Computer Music Journal* 17(2): 23–54.

Roads, C. 1979. "Grammars as Representations for Music." *Computer Music Journal* 3(1):48–56. Also in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press, pp. 443–446.

Rothstein, J. 1992. *MIDI: A Comprehensive Introduction*. Madison, Wisconsin: A-R Editions.

Schottstaedt, B. 1992. "Common Lisp Music." Unpublished software document, Center for Computer Research in Music and Acoustics. Stanford, California: Stanford University,

Smaill, A., and G. Wiggins. Forthcoming. "Hierarchical Music Representation for Analysis and Composition." *Computers and the Humanities*.

Taube, H. 1991. "Common Music: A Music Composition Language in Common Lisp and CLOS." *Computer Music Journal* 15(2):21–32.

Taube, H. 1992. "Stella: Persistent Score Representation in Common Music." In *Proceedings of the Tenth European Conference on Artificial Intelligence—AI and Music Workshop*. Vienna.

Todd, P. M., and D. G. Loy, eds. 1991. *Music and Connectionism*. Cambridge, Massachusetts: MIT Press.

Winograd, T. 1972. *Understanding Natural Language*. Edinburgh: Edinburgh University Press.

Xenakis, I. 1992. *Formalized Music: Thought and Mathematics in Composition*. Harmonologia Series. Stuyvesant, New York: Pendragon Press.

References

TTrees: A Tool for the Compositional Environment
Glendon Diener
Computer Music Journal, Vol. 13, No. 2. (Summer, 1989), pp. 77-85.
Stable URL:
http://links.jstor.org/sici?sici=0148-9267%28198922%2913%3A2%3C77%3ATATFTC%3E2.0.CO%3B2-2

The Interim DynaPiano: An Integrated Computer Tool and Instrument for Composers
Stephen Travis Pope
Computer Music Journal, Vol. 16, No. 3. (Autumn, 1992), pp. 73-91.
Stable URL:
http://links.jstor.org/sici?sici=0148-9267%28199223%2916%3A3%3C73%3ATIDAIC%3E2.0.CO%3B2-F

Machine Tongues XV: Three Packages for Software Sound Synthesis
Stephen Travis Pope
Computer Music Journal, Vol. 17, No. 2. (Summer, 1993), pp. 23-54.
Stable URL:
http://links.jstor.org/sici?sici=0148-9267%28199322%2917%3A2%3C23%3AMTXTPF%3E2.0.CO%3B2-M

Grammars as Representations for Music
C. Roads; Paul Wieneke
Computer Music Journal, Vol. 3, No. 1. (Mar., 1979), pp. 48-55.
Stable URL:
http://links.jstor.org/sici?sici=0148-9267%28197903%293%3A1%3C48%3AGARFM%3E2.0.CO%3B2-2