Surveying Musical Representation Systems: A Framework for Evaluation

Geraint Wiggins[⋆†], Eduardo Miranda[⋆†],
Alan Smaill[⋆], Mitch Harris[⋆]
Department of Artificial Intelligence[⋆], Department of Music[†]
University of Edinburgh
Edinburgh, Scotland.
Email:{geraint,E.Miranda,A.Smaill,M.Harris}@edinburgh.ac.uk

Abstract

We provide a framework for the description of music representation systems in terms of two dimensions, which we call *Structural Generality* and *Expressive Completeness*. This allows us to give some criteria for the evaluation of such systems, dependent on the aims of the user of the system. We then survey a range of current representation systems and their implementations in the light of the our characterisation.

1 Introduction

1.1 The Evaluation of Music Representation Systems

We seek to 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 of such purposes, but we can distinguish three general sorts of task.

1. Recording—Here the user wants a record of some musical object, to be retrieved at a later date. Accuracy is the prime concern.

2. Analysis—Here the 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.

3. Generation/Composition—Here 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 is needed.

Our classification below is intended to cover these three situations.

1.2 Structure of the Paper

This paper proceeds thus. After an outline of the concepts we propose for consideration and of the background ideas involved, we discuss the possibility of using a general-purpose representation language developed in Artificial Intelligence, not specifically targeted at musical applications. We then consider a representative selection of music representation systems in the light of the discussion in the introduction, above, and in the next section. We restrict our attention to systems that work on the level of notes and more abstract structures, rather than systems for dealing with (say) description of timbre.

Some systems raise additional 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 paper concludes by summarizing why we consider this analysis of music representation systems to be useful.

2 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 easily to capture abstract musical content; 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.

The diagram below shows several well-known representation systems classified according to their position with respect to the 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.

2.1 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 sub-parts, 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 75, pp109-117].

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 rather 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 "re-interpretation"; 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.

2.2 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, C) require us to state *how something is to be done*. On the other hand, declarative languages (*e.g.,* Lisp, 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 might 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

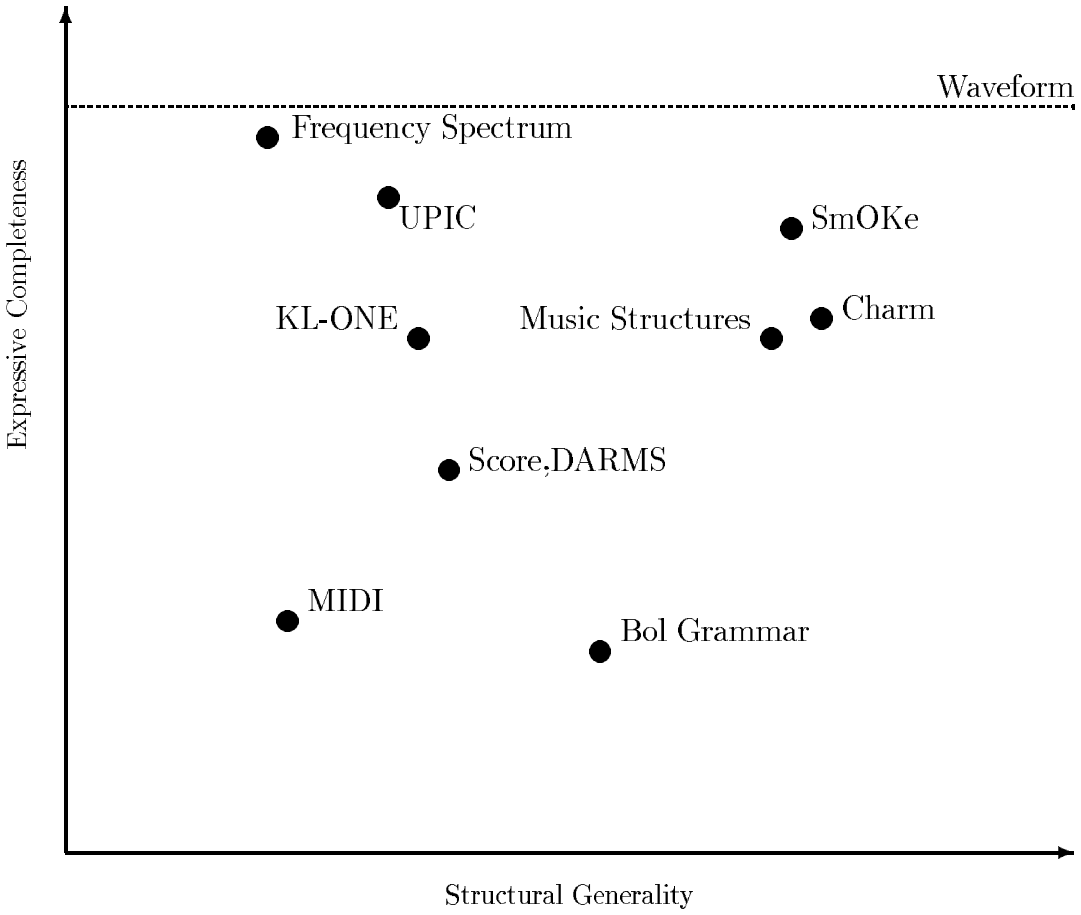


Figure 1: Two Dimensions for Comparison of Music Representation Systems

languages, like 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? 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 *vs.* 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 can 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 on to 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.

2.3 General Purpose Representation Systems

Before looking at purpose-built 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 regime 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 & Levesque 85].

KL-ONE [Brachman & Schmolze 85] has been used as the basis for a musical application [Camurri *et al* 92]. 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 can 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 of musical objects, or are there special characteristics involved that raise special problems?

Some musical systems privilege the notion of time when describing musical objects—see [Balaban 88, Diener 88]. 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 Messiaen's "Modes d'intensités et valeurs" illustrate how time and pitch dimensions (among others) may 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 where 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 look at later. Apart from this, however, the problems raised in music representation are not so different from the general case.

3 Evaluating Music Representation Systems

3.1 Introduction

This section is a survey of specialized music representation systems, evaluated in terms of our proposed two dimensions. There are many more systems than we can possibly cover here, so we have tried to choose a representative member for each subclass. 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 two-fold. 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.

3.2 Spectrum Analysis

We have already mentioned the raw 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 or phase vocoder (see *e.g.,* [Vercoe 91]), applied to the raw waveform we mentioned before. The result of the analysis is a three-dimensional mathematical structure, graphable as frequency *vs.* time *vs.* amplitude. The difference between this representation and the raw waveform is that the individual partials are separated out from the inscrutable (and so structurally non-general) original wave. It is now much easier to isolate, e.g., 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 edition than it was before, in a form that is musically more meaningful.

Further, if we have a perfect spectrum analyzer, and a perfect additive synthesiser 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.

3.3 MIDI

The MIDI system [Rothstein 92] is a combined hardware and software communications protocol for music electronics. Recent developments of the idea include

definitions for "MIDI Files" that contain all the data required to play a "song" on a particular synthesiser or other MIDI instruments. Some instruments have "System Exclusive" commands to allow 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 identifier (NOTE ON or NOTE OFF), a note number (from a standard numbering of the piano keyboard), and a velocity (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 quite 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 and end time are certainly more structurally general than the spectrum representation, there is no further allowance for structural annotation until we reach the level of "MIDI File", which is intended to encapsulate a whole "song". The pitch dimension is equally homogeneous. Thus, the kind of multi-level and multi-view representations discussed below are impossible.

3.4 Score Representations: DARMS

For completeness, we must cover a computer scoring system, which, like any scoring system, can be (ab)used as a representation for musical objects. One such is DARMS [Erickson 75]. DARMS is a language that allows representation of traditional Western scores in the way considered most computer-friendly at the time of its inception—*i.e.*, 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 some-

where 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 our diagram as the score itself.

3.5 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, non-graphical 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 non-graphical ones.

A good example of graphical representation systems is that used in the "UPIC", a computer musical instrument intended to embody the some of the ideas in [Xenakis 71]. 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 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 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 quite well on expressive completeness, due to the high granularity and flexibility of its basic concept of "note", it is only slightly more structurally general than MIDI.

3.6 Music Programming Languages

3.6.1 Introduction

Programming languages, specialized or otherwise, are increasingly being used for tasks related with 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 that we consider carefully the contribution of the host language when we evaluate 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* on our diagram, because they are different in kind from the representation systems shown there. Nevertheless, it is possible to discuss the representations they use.

3.6.2 The Music-N Family

The Music-N family began at AT&T Bell Laboratories [Mathews 69]. It has spawned descendants such as Music V, Music 11, Csound [Vercoe 91], and, recently, CLM [Schottstaedt 92].

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 and pitch, and maybe some 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 (re)produce a musical object very accurately. However, structural generality is less satisfactory: notes may be grouped in sections, but we cannot specify relationships between sections, nor parameterize the sections themselves. No hierarchical arrangement is possible beyond the "section" level.

3.6.3 Common Music

Common Music [Taube 92a] is a language that allows one to write programs that generate sequences of notes in a variety of formats—*e.g.,* MIDI events, or Csound notes. It is a descendant of PLA and SCORE of [Loy & Abbott 85], 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 (*i.e.,* execution, since Lisp is a functional language), 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 datatypes, and so 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 generality: 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.

3.6.4 Stella

Stella [Taube 92b] uses a frame-like representation [Minsky 81], enhancing Common Music to admits 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: *e.g.,* 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.

3.6.5 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, while

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.

3.7 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 sub-parts. For example, in English, a sentence can be composed of a noun phrase followed by a verb phrase. This is often written:

$$\text{Sentence} \longrightarrow \text{NounPhrase VerbPhrase}$$

to form, along with definitions for other syntactic structures, a "Phrase Structure Grammar". Other styles of grammar exist, such as "Categorial Grammars", where 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\NP; then, given the rule

$$X + Y\backslash X \longrightarrow Y$$

the two 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 hard, according to its expressive power [Winograd 72].

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 paper.

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—many of the same issues arise. The "Bol Processor", BP1, [Bel & Kippen 92] is a good example of a grammar-based system. It generates pieces of music ("qa'idas") for "tabla" drumming; according to expert evaluators, its output is credible. BP1 works from a top-down, phrase-structural analysis of the qa'ida form, sub-dividing 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 & Kippen define a form of grammar called a pattern grammar, with 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 85] is a good survey of grammar-based representations. [Lerdahl & Jackendoff 83] is a detailed discussion of a particular system.

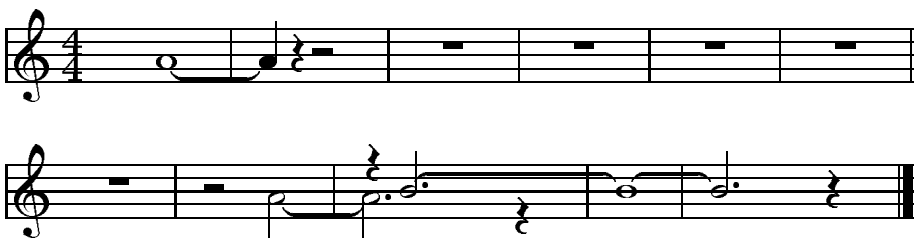
3.8 Music Calculi: Balaban's "Music Structures"

A good example of the notion of a "music calculus" is Mira Balaban's "Music Structures" [Balaban 92]. The point of designing a calculus for knowledge representation is that one would like a language for that representation, which 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", $\bullet$, which is essentially the set insertion operator.

This yields a very powerful and flexible representation system, which is too complicated to explain in detail here. Instead, we give an example (borrowed from [Laske *et al* 92]). Consider the following music structure, presented in a simplified form that treats $\bullet$ like "cons" in Lisp, so (ms_1 $\bullet$ (ms_2 $\bullet$ NIL)) is written as (ms_1 ms_2). The music structure

([a,5]@-20 ([b,10]@5 [a,5]@2)@8)

represents this sequence of notes (rests are implicit; we have added the time signature and barlines 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:

([a,5]@-20 [b,10]@13 [a,5]@10)

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

We can assign names to music structures, and, because the time-stamp operator is relative, use the same structure many times in (say) a motivic piece of music. 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" — *viz.,* arbitrary labelings attached to music structures.

One advantage of Balaban's system is its open-endedness. The symbols used are mostly freely user-chosen, and so are open to free interpretation—though this can lead to the construction of *ad hoc* operators, such as Balaban's "∼" (overlap horizontal concatenation) operator. This extensibility means that the system can have high expressive completeness, qualified by the comment that it is in a sense by default, because there is no an *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 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 (*e.g.*) different tuning systems, is the use of the real line as the representation of time. While we believe Balaban

is correct to state that the required mathematical properties of time are those of the real numbers, it is rarely the case that musicians 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.

3.9 Object Orientation: SmOKe

We outlined the basic notion 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 how it relates to the other systems covered here, and to our evaluation parameters.

The SmOKe (Smallmusic Object Kernel) system of [Pope & *et al* 92] 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, in terms of class hierarchies of objects. Objects "share state and behavior and implement the description language as their protocol". Implementation of SmOKe is explained in [Pope & *et al* 92] *via* the object-oriented SmallTalk-80 language.

Many of the wide-ranging and powerful capabilities of SmOKe are outside the scope of our example evaluation for this paper. 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 synthesisers or music programming languages. It also admits scores, including traditional Western score notation.

Implementations of SmOKe fare rather 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 available. 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, will 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—therefore,

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).

3.10 Abstract Representation: CHARM

The *Charm* system (Common Hierarchical Abstract Representation for Music) [Harris *et al* 91] 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 a 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 is 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 catered 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 *et al* 91]. *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—but it is left to the designer or user of a given concrete representation to build the necessary implementation. Then, 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 here 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, in [Smaill *et al* 90], we explain how an analysis program designed for the twelve-tone scale was used on quarter-tone music *without changing the code of the analyzer.* 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 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 formulæ 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 motif). 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 formulæ in the constituent specification, the structural generality of *Charm* is very high, since any property of or relationship between constituents can be represented explicitly.

4 Symbolic and Sub-symbolic 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 employing parallel distributed processing (PDP, or connectionist) [Leman 88, Leman 91].

While 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 hypothesised representations or processes below it are sub-symbolic with respect to our abstraction boundary. PDP representations might well be appropriate (*e.g.,* for identifying musical events). Furthermore, PDP systems are not intrinsically sub-symbolic—it 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 such.

5 Conclusions

The main thrust of this paper has been about the representational adequacy of different approaches to music representation. As researchers in Artificial Intelligence

and Cognitive Psychology have been only too 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 upon 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. While 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 and are first steps towards 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 place the emphasis on 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.

6 References

[Balaban 88] M. Balaban. A music-workstation based on multiple hierarchical views of music. In C. Lischka and J. Fritsch, editors, *14th International Computer Music Conference*, pages 56–65. Computer Music Association, 1988.

[Balaban 92] M. Balaban. Music structures: Interleaving the temporal and hierarchical aspects in music. In O. Laske, M. Balaban, and K. Ebcioglu, editors, *Understanding Music with AI – Perspectives on Music Cognition*, pages 110–39. MIT Press, Cambridge, MA, 1992.

[Bel & Kippen 92] B. Bel and J. Kippen. Bol processor grammars. In O. Laske, M. Balaban, and K. Ebcioglu, editors, *Un-*

derstanding Music with AI – Perspectives on Music Cognition, pages 366–401. MIT Press, Cambridge, MA, 1992.

[Brachman & Levesque 85] R.J. Brachman and H.J. Levesque, editors. *Readings in Knowledge Representation.* Morgan Kaufmann, California, 1985.

[Brachman & Schmolze 85] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[Camurri *et al* 92] A. Camurri, M. Frixione, C. Innocenti, and R. Zaccaria. A model of representation and communication of music and multimedia knowledge. In B. Neumann, editor, *Tenth European Conference on Artificial Intelligence*, pages 164–8, Vienna, 1992. John Wiley and Sons, Chichester, England.

[Diener 88] G. Diener. Ttrees: an active data structure for computer music. In C. Lischka and J. Fritsch, editors, *Proceedings of the 14th International Computer Music Conference*, pages 184–88. Computer Music Association, 1988.

[Erickson 75] R. F. Erickson. The DARMS project: A status report. *Computing and the Humanities*, 9(6):291–298, 1975.

[Harris *et al* 91] M. Harris, A. Smaill, and G. Wiggins. Representing music symbolically. In *IX Colloquio di Informatica Musicale*, Genoa, Italy, 1991.

[Laske *et al* 92] O. Laske, M. Balaban, and K. Ebcioglu. *Understanding Music with AI – Perspectives on Music Cognition.* MIT Press, Cambridge, MA., 1992.

[Leman 88] M. Leman. Symbolic and subsymbolic information processing in models of musical communication and cognition. *Interface*, 18:141–60, 1988.

[Leman 91] M. Leman. Tonal context by pattern integration over time. In M. Leman, editor, *IX Colloquio di Informatica Musicale*, Genoa, Italy, 1991.

[Lerdahl & Jackendoff 83] F. Lerdahl and R.S. Jackendoff. *A Generative Theory of Tonal Music.* The MIT Press, Cambridge, MA., 1983.

[Loy & Abbott 85] G Loy and C. Abbott. Programming languages for computer music synthesis, performance and composition. *Computing Surveys*, 17:235–265, 1985.

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

[Minsky 81] M. Minsky. A framework for representing knowledge. In J. Haugeland, editor, *Mind Design*, pages 95–128. MIT Press, Cambridge, MA, 1981.

[Nattiez 75] J.-J. Nattiez. *Fondements d'une sémiologie de la musique.* Union Générale d'Editions, Paris, 1975.

[Pope & *et al* 92] S. T. Pope *et al.* The Smallmusic Object Kernel: A music representation, description language, and interchange format. Anonymous ftp from ccrma-ftp.stanford.edu:/pub/st80, 1992. N.B. draft only.

[Roads 85] C. Roads. Grammars as representations for music. In C. Roads, editor, *Foundations of Computer Music*, pages 443–46. MIT Press, Cambridge, MA, 1985.

[Rothstein 92] J. Rothstein. *MIDI : a comprehensive introduction.* Oxford University Press, Oxford, 1992.

[Schottstaedt 92] B. Schottstaedt. Common Lisp Music. Unpublished software document, CCRMA, Stanford University, California, 1992.

[Smaill *et al* 90] A. Smaill, G. Wiggins, and M. Harris. Hierarchical music representation for analysis and composition. In *Proceedings of the Second International Conference on Music and Information Technology*, Marseilles, France, 1990. To appear in Computers and the Humanities, Flushing, New York.

[Taube 92a] H. Taube. Common music. Technical report, ZKM, Germany, 1992.

[Taube 92b] H. Taube. Stella: persistent score representation in common music. In G. Widmer, editor, *10th ECAI: AI and Music Workshop*, Vienna, 1992. ECCAI.

[Vercoe 91] B. Vercoe. *The Csound Reference Manual.* Addison-Wesley Publishing Company, Cambridge, MA, 1991.

[Winograd 72] T Winograd. *Understanding Natural Language*. Edinburgh University Press, Edinburgh, 1972.

[Xenakis 71] I. Xenakis. *Formalized Music*. Indiana University Press, Bloomington, Indiana, 1971.