Representing Music Symbolically

Mitch Harris, Alan Smaill and Geraint Wiggins [*]

Abstract

The automated understanding and generation of music is an area of research that raises problems that are central to the Artificial Intelligence enterprise. Recent work at Edinburgh has aimed to use a symbolic AI approach for this field. We indicate how a more abstract understanding of music representation unifies this approach, and how logical descriptions of hierarchical structures can be incorporated. We relate this work to alternative approaches, describe some projects carried out in this framework, and indicate planned future work.

Keywords: Key; Knowledge Representation; Structured Hierarchy; Abstract Data Type.

1 Introduction

…les calculs n'ont cessé d'accompagner la musique tout au long de son histoire, et dans toutes les civilisations …
([Boulez 85, p 73])

Musicians have an intuitive awareness of the depth of possibilities of musical structure, and how these structures can be manipulated to create new forms of musical understanding. At first sight, computers would seem to be an ideal tool for exploring this mode of expression, yet there seems to be a barrier of *form* between musical ideas and structures that can be implemented on a computer. Our aim is to give ways to overcome this barrier.

There has been relatively little work in the area of AI and music, yet the problems it poses are central to the AI enterprise. There is a particular problem for

[*]Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN; Email:{M.Harris,A.Smaill,geraint}@uk.ac.edinburgh

the traditional symbolic AI approach, as we do not have the benefit of the sort of well developed syntactic and semantic analyses that have been worked out for natural language, for example. On the other hand, while the complementary sub-symbolic approach is effective, and perhaps necessary, for some musical tasks (see section 2.3), such solutions will not give to the user of an AI system the sort of structured understanding and manipulative ability that is usually wanted.

In what follows, we describe how we have attempted to organise formal systems for music representation for manipulation in an automated system. Our intention is to allow a wide range of possible representations, while still permitting the sharing of software, through the use of *abstract data types*. Thus the power of automation may be made available to composers and analysts, by allowing them to think of and access music in terms familiar to them, and yet which also allow automated manipulation.

In section 2, we outline our notion of representation. In section 3, we indicate how an *abstract* approach helps us to deal uniformly with different sorts of representation. In section 4, the topic of higher-level musical structures and their specification is introduced. After describing in section 5 three more extended pieces of work carried out in the style we advocate, we summarise our conclusions and indicate future research directions.

2 What is there to represent?

Before we describe our representation, let us establish what it is that we wish to represent. There are various possibilities, so we will consider them in turn.

2.1 Score / Performance

Often musical databases hold transcriptions of scores, and are intended to record the information in the score rather than to describe some music more directly. The difference can be fairly small for written music in (*eg*) the classical western tradition, but for many other styles the score (when it exists) relates less closely to music as performed or experienced. Therefore we choose to represent in the first instance music *as performed*. This means, for example, that a score that does not determine the order of some musical events will correspond to several possible performances and so to a collection of representations.

2.2 Physical / Psychological

The process of forming abstractions nearly always has a psychological component. At the lower levels the psychophysics of perception are important; for example the categorical perception of pitch intervals makes possible the abstraction of pitch into note names. Very similar effects occur in our perception of speech; the phoneme *t* does not correspond to any unique pattern of sound energy as this may vary significantly depending on the preceding and subsequent vowel sounds and the position of the consonant in a word. In general, there is no absolute mapping between any physical event and a musical perception, because of the effects of context, categorical perception and the various kinds of filtering which occur due to our biological and neurological make up. See [Handel 89] for discussion of these and related issues.

Having said this, phonemes are a useful abstraction in linguistics, and similarly pitches and durations are useful abstractions for music to use as a base-line for symbolic representation, although the ambiguity between the physical and the psychological is inherent and sometimes problematic. At higher levels, similar considerations will also apply, and either the psychological or the physical phenomena may be the objects of interest.

2.3 Symbolic / Subsymbolic

Having decided on an abstraction boundary for our representation (in terms of events with pitches and durations) we can construct formal systems above this layer. One example of such a system is Western tonal music, though this is too restrictive in general. Below this layer we are, by our definition, *outside* the formal system.

To make a mapping between the physical world and the perception of a musical note we need a *subsymbolic* system. This, in turn may be abstractly represented, although it may be argued that a discrete symbol system is inadequate for this and analogical representations or connectionist systems will be needed ([Leman 88]).

2.4 Analysis / Generation

Our use of a basic level of representation on the level of note-like events is similar to Jackendoff's notion of a "musical surface" [Jackendoff 87, p 217]. But of course this is only a starting point – we want to be able to talk about groupings of these basic events, according to whatever criteria interest us, and about groupings of these groupings, and so on.

We draw a distinction between the unadorned description of the events that make up a piece of music, and whatever richer description interests the listener or composer. Thus on top of the basic description we introduce a mechanism that allows such higher order structures to be introduced in a way appropriate to the task at hand. For example, the analyst may wish to determine metrical or tonal information that is not explicit in the description of the notes of the piece. The mechanism we propose allows the user to represent such higher-level information, either by asserting it directly, or by indicating in general terms the conditions under which these higher-level properties hold. For example, traditional harmonic analysis can be thought of as a series of rules for determining harmonic progressions for music written in a certain style; as such, the rules could be used to indicate what harmonic progression is suggested by given notes, and this could be determined automatically.

In the terms of Nattiez's description in [Nattiez 75], our basic representation is on the level of the "niveau neutre" – it certainly makes no attempt to represent any compositional or affective features that the music may have. However, the mechanism for higher-order structures that we describe in section 4.1 allows the composer to manipulate musical material using structuring devices of the composer's choice, just as it allows a variety of interpretations of the piece to be expressed.

2.5 Representation / Implementation

In what follows we will propose some data structures which we believe are generally useful for reasoning about pieces of music, and which should allow sharing of algorithms for implementations that respect our representation principles. Our proposal is thus *not* of a new software system; indeed, we have implemented parts of this work in several different versions, and in two languages. We envisage that different users will want to describe even notions as basic as pitch and duration in different ways; our proposal is designed to allow this.

3 Abstract Representation

Anyone who looks at computer systems for music representation is quickly struck by the variety of systems on offer. There are some attempts to standardise; for example relatively low-level information may be encoded using MIDI, DARMS *etc* communications protocols. However the possibilities are enormous, and the diversity often prevents us from using software written for one representation for another representation.

Computer scientists have come across this problem in many areas, and have developed a response to it in the use of *abstract data types.* The main idea is that there are patterns of operations over data that recur frequently in different *concrete* implementations. If the operations are expressed in terms of these general patterns, then they can be made to apply to whatever particular implementation is used, provided we know how to relate the concrete implementation to the *abstract* patterns.

For example, if we consider an operation such as determining the interval between the pitch of two notes, this will be performed in one way if the pitches are represented in Hertz, in another if they are given in number of semitones above middle C, in another if they are given as traditionally notated on a stave, and there are many other possibilities. If we regard all of these as instances of the same computation at an abstract level, this will be one of the operations of our abstract data type. We define such an abstract data type for the descriptions of collections of notes in general. We call this the *basic representation*. This is to be contrasted with particular instantiations of this basic representation in some *concrete* data type.

Once we move beyond this basic representation, there are even more possible operations the user may wish to perform, for example generating music by repeated transformations of some structure that is never explicitly heard (as in "Traum A", section 5.3). We will use the notion of *constituent* to describe in general terms higher-level musical structures that might be so generated, or appear in some analytic process.

We now describe our proposal more formally. The reader not interested in the technical details may skip to section 5.

3.1 The Basic Representation – Specification

The internal structure of our (simplified) event representation is as follows. We define abstract datatypes to represent Pitch (and Pitch Interval), Time (and Duration), Amplitude (and Relative Amplitude) and Timbre, which we will not discuss here – it will be the subject of future work. The abstract event representation is then the cross product

$$\text{Pitch} \times \text{Time} \times \text{Duration} \times \text{Amplitude} \times \text{Timbre}$$

The abstract data type for Time is formed as follows. (The others, except for Timbre, are the same modulo renaming.) The objects of interest are points in time and intervals between them (durations). We want to be able to compute durations from pairs of times, durations from pairs of durations, and so on. We therefore require the existence of functions $\text{add}_{\text{x}y}$ and $\text{sub}_{\text{x}y}$ where x and y are each one of { t d }, standing for Time and Duration, respectively, defining the types of the arguments. The type of the result is unambiguous. Thus, we have functions

$$\begin{array}{ll} \text{add}_{\text{dd}}\text{:} & \text{Duration} \times \text{Duration} \rightarrow \text{Duration} \\ \text{add}_{\text{td}}\text{:} & \text{Time} \times \text{Duration} \rightarrow \text{Time} \\ \text{sub}_{\text{tt}}\text{:} & \text{Time} \times \text{Time} \rightarrow \text{Duration} \\ \text{sub}_{\text{dd}}\text{:} & \text{Duration} \times \text{Duration} \rightarrow \text{Duration} \end{array}$$

There is an ordering on the type Duration, $\leq$. A distinguished symbol for the zero duration, the operation add_{dd} and an inverse are defined so that they make

Duration a linearly ordered commutative group. Formally, this means that duration is a Generalised Interval Structure in the sense of [Lewin 87], with extra properties.

The abstract data types for Pitch and Pitch Interval, and for Amplitude and Relative Amplitude, may be derived by appropriate renaming of the properties in the above description.

3.2 The Basic Representation – Implementation

Having defined our abstract data type, we now indicate what we mean by the implementation of the type via the provision of corresponding concrete operations.

Each member of a *concrete* event structure is associated with a unique Identifier, for efficient reference by software routines. Such reference is made via destructor functions on the datatypes. We require that the following destructor functions be defined in any instantiation of our representation.

Get$X$ where X is one of { Pitch Time Duration Amplitude Timbre } are unary functions which return the appropriate component of the event tuple associated with the identifier given as their arguments

PutEvent is a function taking an event tuple as its single argument and returning the identifier associated with it

Note that these functions (and those for constituents, specified in Section 4.1) are normally used in conjunction with a database in which the events and constituents making up an idealised performance are stored. The precise form of such a database is immaterial to applications using this representation because of the availability and transparency of the destructor functions.

3.2.1 Example

An example of the event structure, implemented here as a neutral logical term, might use the instances of the abstract data types shown in Figure 1. Timbre is not considered here – we will avoid the issue with ellipsis (...). We associate each event in a represented score with a unique identifier – in this example of the form $\mathsf{e}\mathsf{N}$ where N is in Integer Number. Finally, in this instance of the representation we notate the existence of the association between the identifier and the appropriate element of the type of data type tuples by the ϵ relation.

Figure 2 shows the specification in the event structure of the first twelve notes of Webern's Variations for Piano, Op 27, as in Figure 3. Note that we do not intend to suggest that this is a readable or person-friendly notation. The attempt

Pitch	=	{ a b c d e f g } × { ♮ ♯ ♭ } × Integer Number representing traditional Western notation – note name, accidental, and octave
Pitch Interval	=	Integer Number representing integer numbers of semitones
Time	=	Integer Number × Integer Number representing rational numbers of crotchet beats
Duration	=	Integer Number × Integer Number representing rational numbers of crotchet cbeats
Amplitude	=	Integer Number × Integer Number representing rational numbers of decibels (dB)
Relative Amplitude	=	Integer Number × Integer Number representing rational ratios of Amplitude

Figure 1: An example event instantiation

is to supply a *standard* (abstract) notation, which may then be translated to user-friendly form *in a general way.*

ϵ(e00, ⟨f,♮, 4⟩, 1/4, 1/2, 10/1, ...)
ϵ(e01, ⟨e,♮, 5⟩, 1/4, 1/2, 10/1, ...)
ϵ(e02, ⟨b,♮, 3⟩, 1/2, 1/4, 10/1, ...)
ϵ(e03, ⟨f,♯,3⟩, 3/4, 1/4, 10/1, ...)
ϵ(e04, ⟨g,♮, 4⟩, 3/4, 1/4, 10/1, ...)
ϵ(e05, ⟨c,♯,5⟩, 1/1, 1/4, 10/1, ...)
ϵ(e06, ⟨a,♮, 2⟩, 3/2, 1/2, 10/1, ...)
ϵ(e07, ⟨b,♭, 3⟩, 3/2, 1/2, 10/1, ...)
ϵ(e08, ⟨e,♭, 4⟩, 7/4, 1/4, 10/1, ...)
ϵ(e09, ⟨c,♮, 4⟩, 4/2, 1/4, 10/1, ...)
ϵ(e10, ⟨d,♮, 5⟩, 4/2, 1/4, 10/1, ...)
ϵ(e11, ⟨g,♯,4⟩, 9/4, 1/4, 10/1, ...)

Figure 2: Webern, Op 27, bars 1-3

4 Hierarchical representation

It is widely agreed that a system to manipulate musical representations must allow higher level structures to be introduced hierarchically [Balaban 88, Buxton & *et al* 78]. On the other hand, it is equally widely agreed that such

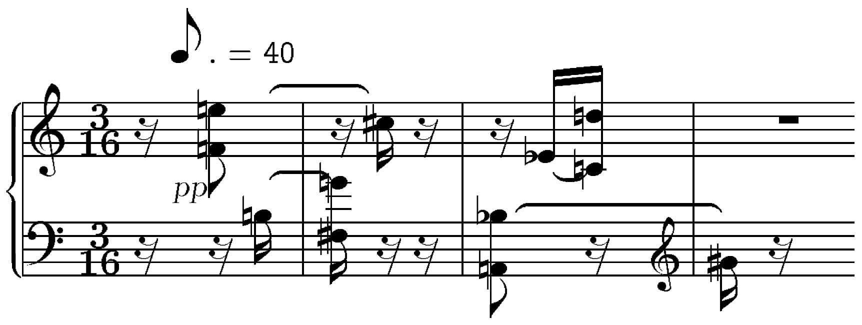

Figure 3: Webern, Op 27, bars 1-4

structures and the groupings or kinds of groupings which they delineate must not be imposed on the user of a representation system. Further, it must be possible to assign different hierarchical structures simultaneously to a given set of musical events, in order, for example, that different possible structural interpretations may be represented where there is ambiguity, or that different kinds of information may be represented and related together but kept logically separate (*eg* information about harmony and orchestration).

4.1 The Constituent Representation – Specification

In our representation, we use *constituents* to delineate groupings of events and other constituents. A constituent, at the abstract level, is a pair of the form

$$\langle \text{Properties/Definition}, \text{Particles} \rangle$$

Properties/Definition allows logical specification of the relationships between the Particles of this constituent in terms of membership of certain classes, which may be defined externally by the user; the reason for the two-part name for this component will become clear below

Particles is the set of the events and sub-constituents making up this constituent.

A sub-constituent of a constituent is one of its Particles or a sub-constituent of one of them. The Particles of a constituent are restricted so that no constituent may be a sub-constituent of itself. Thus, the constituent structure of a performance in our representation is a directed acyclic graph.

4.1.1 Logical Classification of Constituents

For constituents to be maximally useful, we require that their properties be defined in such a way as to be easily and efficiently available for testing and/or manipulation. That is, given that the user should able to specify whatever musical groupings s/he is interested in, it must be possible to indicate that these groupings have some structural properties – for example, that they are constituted horizontally or vertically with respect to time.

In order to specify Properties more generally, but still allow efficient inference from them, we propose the following approach. The Properties component of a constituent is a pair,

$$\langle \mathit{spec}, \mathit{environment} \rangle$$

where *spec* is a logical specification, and *environment* is a (possibly empty) set of values for the result of GetPitch *etc*, when applied to this constituent. The idea is that the *spec* is a logical specification for the defining property of the constituent, which can be checked by looking at the particles of the constituent; the *environment* supplies event-like information on Pitch, Time *etc* associated with the constituent directly.

First, we define the specification language for *spec*s. We use a first order logic, with the conventional connectives. We already have the destructor functions on structures and datatypes, and the arithmetic and comparison functions described before (and their derivable relations). Then we can naturally specify the property of (*eg*) a monophonic line, or *stream*, like this, where p_i range over particles:

$$\begin{aligned} \mathit{stream} \leftrightarrow & \\ \forall p_1.\neg\exists p_2.\ & p_1 \neq p_2 \\ & \wedge \mathrm{GetTime}(p_1) \leq \mathrm{GetTime}(p_2) \wedge \\ & \mathrm{GetTime}(p_2) < \mathrm{add}_{td}(\mathrm{GetTime}(p_1), \mathrm{GetDuration}(p_1)) \end{aligned}$$

This means that the constituent has no particle which starts between the beginning and end of any other particle.

Similarly, we could specify the orthogonal type, the time *slice*, where some point in time is common to every particle in the constituent, like this:

$$\begin{aligned} \mathit{slice} \leftrightarrow & \\ \exists t.\forall p_i.\ & \mathrm{GetTime}(p_i) \leq t \wedge \\ & t \leq \mathrm{add}_{td}(\mathrm{GetTime}(p_i), \mathrm{GetDuration}(p_i)) \end{aligned}$$

4.2 The Constituent Representation – Implementation

The implementation of the constituent is very similar to that of the event. The implementation must be a tuple of the form

$$\langle \mathrm{Identifier}, \mathrm{Properties}, \mathrm{Definition}, \mathrm{Particles}, \mathrm{Description} \rangle$$

Identifier is as in the event implementation;

Properties and Definition are, as at the abstract level, a logical specification of the structural properties of the constituent. At the implementation level, though, we make a distinction between properties specified in terms of externally defined predicates and those defined by the constituent itself. More on this below;

Particles are as in the abstraction above;

Description is an arbitrary structure, defined by the user, which is intended for annotation of useful information. Note that no interpretation is given for this component, and that while software may freely write to it, no software *using* its information in any strong sense can guarantee to be portable.

We require that constituents have appropriate typing and destructor functions, as for events.

4.2.1 Properties and Definitions

What, then, is this distinction between the Properties and the Definition of a constituent?

In checking that a constituent has the properties required for meaningful application of a given algorithm, or, indeed, in checking that the Particles of a constituent actually do have the Properties claimed, it will always be necessary to use definitions of any propositions given in terms other than of the basic connectives in the logic and the functions and predicates of the abstract data types. Thus, a definition, external to the constituent structure itself is required – examples for the *stream* and *slice* were given above.

Now, there may well be properties which a user wishes to state about his/her constituent which are simply true of that constituent by definition – for example, that a constituent comprises the events of a particular piece of music. While it is clearly the case that such definitions could be written in the same way as the definitions of *stream* and *slice*, above, it is equally clear that doing so could be arbitrarily laborious – consider, for example, representing a full-scale symphony in this way: the existence of each individual note would need to be verified. This is doubly undesirable by virtue of the fact that all this work has already been done anyway, in writing down the events and constituent structure in the first place.

It makes sense, therefore, at the level of implementation, to distinguish between those propositions which are *derivably* true of a constituent (*ie* its Properties) and those which are *definitionally* true of it. The Definition component contains the latter. As an example, the constituent representing Debussy's "Syrinx", as discussed in [Wiggins *et al* 89, Smaill *et al* 90], might be represented thus, where c41 is the identifier of this constituent and e1 ... e999 are the events comprising it:

$$(\text{c41}, \langle\text{stream}, \{\}\rangle, \text{syrinx}, \{ \text{e1} \ldots \text{e999} \}, \text{“A solo flute piece by Claude Debussy”})$$

In this way, the property *syrinx* is defined to be true of the constituent, but will not be unnecessarily checked by a specification checker, because it is distinguished as a Definition; nor will the user have to give an external definition for it.

4.2.2 Defaults

We specify defaults for each of the Get$X$ functions over constituents as follows, so that we can be sure that, for example, GetPitch will return *some* value which is meaningful, at least to the user who defined the default. The defaults can then be built in to the implementation of the Get$X$ functions. However, it may well in general be undesirable or impossible to specify a universal default system for any given application, and so we allow the specification of over-riding values in the *environment* component of the Properties pair. The *environment* is a set of pairs of named constants and values. The constants are named after the abstract datatypes of the representation (*viz* Pitch, Time, Duration, Amplitude, Timbre), and the associated values are then returned as the result of any call to the corresponding Get$X$ function with this constituent as parameter. For example, consider a constituent, with Identifier c0, whose Properties are defined thus:

$$\langle\text{stream}, \{\text{Time} = 0\}\rangle$$

This constituent has the stream property defined above, and any call of GetTime(c0) will return the value 0.

4.3 Example

A typical example constituent is given in Figure 4 (c00). It records the fact that the events shown in Figure 2 are related together. That they form the subject of the movement is represented by the Definition; and the underlines _ in the Properties and Description positions indicates there are no entries there. In this instance of the representation, we use the relation κ to express that we are dealing with a constituent. The identifier is of the form cN where N is in Integer Number.

Let us now consider how use of the representation given in Figures 1, and 2 allows us to express different readings of the same notes. From the logical descriptions of the constituents it is possible to check or generate constituents with the properties described.

We have already shown how the initial notes may be represented in our notation. In fact this twelve note constituent has the property of containing each degree of the chromatic scale exactly once - that is, it defines a series. We can define what it is for a constituent to have this property, and make the assertion that the first twelve notes form a series in the following form (figure 4, (c001)). Note that this has the *same* notes as the previous example, but is distinct as a constituent.

Suppose now that we wish to represent some of the internal relations of these notes. For example, if we simply collect together notes sounded simultaneously, we find the four two-note chords we can write as in figure 4 (c03-c06). Then the alternation of two-note chords and single notes can be expressed by a constituent that gathers together the chord constituents and the remaining single notes (c07). Gathering the notes in groups of three in temporal order gives the four triples (c08-c11) (figure 4).

```
κ( c00, _ , subject, { e00 e01 e02 e03 e04 e05 e06 e07 e08 e09 e10 e11 }, _ )
κ( c01, ⟨series,{}⟩, _ , { e00 e01 e02 e03 e04 e05 e06 e07 e08 e09 e10 e11 }, _ )
κ( c03, ⟨chord,{}⟩, _ , { e00, e01 } , _ )
κ( c04, ⟨chord,{}⟩, _ , { e03, e04 } , _ )
κ( c05, ⟨chord,{}⟩, _ , { e06, e07 } , _ )
κ( c06, ⟨chord,{}⟩, _ , { e09, e11 } , _ )
κ( c07, ⟨alternation,{}⟩, _ , { c03, e02, c04, e05, c05, e08, c06, e11 } , _ )
κ( c08, ⟨triple,{}⟩, _ , { e00, e01, e02 } , _ )
κ( c09, ⟨triple,{}⟩, _ , { e03, e04, e05 } , _ )
κ( c10, ⟨triple,{}⟩, _ , { e06, e07, e08 } , _ )
κ( c11, ⟨triple,{}⟩, _ , { e09, e10, e11 } , _ )
```

Figure 4: Constituents for Webern Op 27

Now, the first three of these triples are related in pitch terms – the pitches can be obtained by transposition modulo the octave. This fact can be expressed in a similar way to our other constituents. In this way, the constituent mechanism allows the collection and labelling of events and other constituents.

5 Reasoning with the representation – Some case studies

One approach to Artificial Intelligence in the symbolic tradition aims to simulate "intelligent" behaviour by allowing the goal-directed manipulation of symbolic structures. In presenting our *abstract* basic representation, we have given a *family* of ways of describing music in a neutral manner. In presenting our constituents, we have given ways for users to introduce their own notions of analysis and compositionally interesting features. We now indicate briefly how reasoning takes place via the manipulation of these descriptions so as to permit analysis and generation of music. Reasoning here is thus the inference of higher-level constituent structure during analysis, or of basic structure from generational constituents or from transformations of existing structures.

We now describe some work performed using some of the above ideas. In the first two cases, the use of a particular representation is not central, and our abstract representation framework was used implicitly. In the third case, the representation appears difficult to achieve in other ways, and our representation appears explicitly. [Smaill *et al* 90] describes a further example.

5.1 Case study: Small Scale Analysis

In [Wolte 90], Isabel Wolte designed a program for analysis of musical form in small, simple, classical music (minuets and trios by Mozart and Haydn). Based on earlier work by Steedman ([Steedman 77]), a number of procedures were written to deal with the rhythmic, harmonic, and repetition and similarity information all of which is required for the simplest analysis.

The program consists of three relatively independent parts; each part is responsible for one aspect of the analysis and builds up on the information obtained by the preceding part(s). The musical input is first analysed from a rhythmic point of view: a metre for the given piece is suggested and the input reorganised into bars. Then harmonic constituents are evaluated, and regions of specific keys defined, following the ideas of [Longuet-Higgins 62]. A further analysis of these regions establishes the notion of a phrase, punctuated by a cadence, and provides a simple overall structure of the piece. The analysis is performed bottom-up.

In using our representation for such analysis the musical events which we wish to consider are abstractions of a hypothetical listening process; one which analyses acoustic input and extracts a sequence of pitches and idealised durations (crotchet *etc* or integer numbers of beats) – throwing out (or representing separately) any information about tempo, stress *etc.* A process then acts upon this representation, analysing it to produce constituents representing meaningful units such as cadences. The knowledge of key or time signatures is not assumed and is inferred from the pitch and length information of the musical input.

The program performed well on the restricted range for which it was designed. However, it proved to be brittle outside that range, because of the fixed order of use of the available information. The presence of various experts for different sorts of musical understanding suggests that a blackboard system would work well here, as in [Ebcioglu 88], or a distributed system as suggested by [Minsky 85]. Such systems would share musical representations of the form suggested above.

5.2 Case study: Parsing for Temporal Structure

The ability to determine the internal metrical structure of a piece of music from a performance (or score) with no expressional markers (accents, bar-lines, phrasings *etc*) can be a taught skill – and thus one which seems amenable to formalisation as a parsing process. Where a piece is heard for the first time the parse must happen in temporal order (without lookahead). The inference of higher level metrical structure (time signature and phase) from local events must have a significant 'bottom up' component.

Steedman, in his model of rhythmic analysis [Steedman 73, Wiggins *et al* 89], took the approach of parsing for different metrical feet (*eg* trochees and dactyls) which would only be recognised if they occurred on a strong beat of the metre established up to that point. He had a policy of strict commitment during the parse – the establishment of low level metrical structure always preceded larger (longer) structures, which were never revised once established. This has attractions as a model of listening because it is bottom up and easily implemented as a one-pass process and thus has some psychological plausibility. However, the drawback of sticking to early commitments is that they might turn out to be wrong, leading to misinterpretation of all subsequent structure.

John Whyte has been working on an improved parser which also works in one pass but stores the evidence for different candidate metric structures in a tree, and thus does not suffer from the problem of premature commitment [Whyte 91]. For example, this allows evidence for groupings into two and threes (*hemiola*) to be co-present. There is psychological evidence for such ambiguity ([Handel 89, p 411]). Pitch information can also contribute to this parsing process.

The system acts upon this representation, parsing it to get constituents representing metric chunks. For example, the metrical foot *dactyl* (long-short-short) can be represented as a property thus:

$$\begin{array}{lll}
\textit{dactyl} & \leftrightarrow & \\
 & \exists p_1.\exists p_2.\exists p_3. & \mathrm{add_{td}}(\mathrm{GetTime}(p_1), \mathrm{GetDuration}(p_1)) = \mathrm{GetTime}(p_2) \wedge \\
 & & \mathrm{add_{td}}(\mathrm{GetTime}(p_2), \mathrm{GetDuration}(p_2)) = \mathrm{GetTime}(p_3) \wedge \\
 & & \mathrm{GetDuration}(p_2) = \mathrm{GetDuration}(p_3) \wedge \\
 & & \mathrm{GetDuration}(p_1) > \mathrm{GetDuration}(p_2) \wedge \\
 & & \forall p_4.(p_1 = p_4 \vee p_2 = p_4 \vee p_3 = p_4)
\end{array}$$

This simply expresses the local constraints within the dactyl itself (the relative durations and order of its particles). In practice higher level constituents are needed to constrain the context in which a dactyl is perceived as such, for example, the note which follows a dactyl must be longer than last two notes of the dactyl itself. However, once satisfactory definitions have been created for the necessary "perceived" constituents, we can experiment with different parsing techniques, and the parser will work for any concrete representation. This illustrates how our representation may be applied to the cognitive modelling of musical processes, using constituents to represent perceived structures.

5.3 Case Study: "Traum A"

"Traum A", for solo computer, by Geraint A Wiggins is an algorithmically generated piece, in the minimalist tradition, which shows two ways in which our representation system can be useful to the composer. In particular, the piece requires the simultaneous representation of notes in equal and just temperament scales.

The piece starts from a "seed", which is repeatedly transformed according to a simple rule. The seed is a set of 128 sine waves, initially played simultaneously, at equal amplitude, each an even-tempered semitone from the next, and centred around Middle C. There is an underlying theme, which, though never explicitly heard, directs the transformations as time proceeds. The theme can be thought of as playing throughout the piece – at any given time, one note will be current. In particular, as well as various time displacements which are not so interesting for our purposes, each transformation step performs pitch and amplitude transformations to achieve the following effect.

Initially, we hear a block of noise, which is effectively unpitched. As the piece proceeds, the even tempered semitones tend in amplitude towards values determined by whichever note of the underlying theme is notionally current. If the pitch of a particular tone is close to a harmonic of that theme note (in the usual "power-of-two" harmonic series), the tendency is towards the amplitude of that harmonic; otherwise it is towards zero. When a tone's amplitude gets close enough to that of the corresponding harmonic, its pitch is altered to be exactly that of the harmonic. Thus, it has moved from equal to just temperament (with respect to the current note of the theme). Not only does this produce the obvious effect in the harmony of the piece, but as time progresses, the sound we hear tends towards a single note – at the pitch of the current note of the underlying theme – as the non-harmonic tones fade, and the near-harmonic ones become exactly harmonic. The timbre of that single note is then determined by the particular distribution (in the space of frequencies) of the tones of which it is composed.

Because the underlying theme is expressed in equal temperament, each time we pass from one note to the next, there are significant changes in the relative frequencies of the current and target sounds. This gives rise to a constant shifting of the harmonic spectrum, which provides the interest of the piece.

The creative aspect of the piece, then, lies primarily in the choice of theme and of the values determining "closeness" in pitch and amplitude. Similar variables appear in the transformation of the time displacements of each note.

5.4 Representing “Traum A”

There are two ways in which our representation is able to help us with this piece. The first is rather prosaic, and would presumably be possible in the most basic of representations – the obvious way to represent the set of sine waves produced by each step in the transformation process is as events grouped in constituents. Then the transformation step can be represented as a function from constituents to constituents. The resulting constituents can then be bundled together in one constituent representing the whole piece.

More interestingly, we can define a Pitch data-type which has the ability to represent both equal and just temperament at the same time. We require Pitch and Pitch Interval to be defined as in Figure 5. The intuition is that we represent Pitch as some equal temperament base pitch – defined by the first three components of the tuple: note name, accidental, and octave number, as before – with a harmonic multiple and an octave shift (applied *after* the harmonic multiple) to give us access to the just temperament scale related to each equal tempered note in the type. Pitch Interval is then an integer number of semitones (the first component) and a rational multiple which gives the adjustment between the different scales.

Having made these definitions, and supplied the appropriate destructor functions, we can easily supply the seed, values for the two closeness measures, and the theme (which can also be expressed as a constituent). The implementation of the algorithm itself is then almost trivial.

$$\boxed{\begin{array}{l}\text{Pitch} = \{\ \mathrm{a\ b\ c\ d\ e\ f\ g}\ \} \times \{\ \natural\ \sharp\ \flat\ \} \times \text{Integer} \times \text{Integer} \times \text{Integer} \\ \text{Pitch Interval} = \text{Integer} \times \text{Integer} \times \text{Integer}\end{array}}$$

Figure 5: Pitch/Pitch Interval datatype for “Traum A”

The whole has been implemented in Prolog, and interfaced to the CSound music programming language via completely general routines based on the abstract representation.

This ease of representation, we claim, would not be available in more conventional notations. In particular, the attempt to notate this piece in conventional score notation would be doomed to failure, unless the composer resorted to the addition of indication *for each and every note* whether it were in equal or just temperament. What is more, the compositional process itself is aided by the expression of the just tempered scales with respect to bases in the equal tempered scale. This, again, would be difficult and complicated to represent in more conventional terms.

6 Discussion

It is now appropriate to reconsider how our work fits into the broader context of music representation in general. We have already discussed the problem of 'what to represent' in terms of the various dichotomies of score vs performance, physical vs psychological, symbolic vs subsymbolic, analysis vs generation and representation vs implementation. Clearly, there is no panacea, but it is useful to consider the relative merits of different systems along two dimensions — *expressive completeness* and *structural generality*.

'Expressive completeness' simply refers to the range of raw musical data which can be represented, and 'structural generality' refers to the range of high level structures which 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; even another 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.

Graphical representations of energy spectrum against time (*eg* Xenakis's UPIC) are more useful to musicians than a raw energy waveform but (as yet) there is no uniform way of interfacing these to more abstract structures, particularly those which are not localised in time (*eg* recapitulation). Similar considerations also apply to symbolic representation schemes which give time a privileged status (*eg* [Diener 88, Balaban 88]). However, within the time domain these systems have more *structural generality* than purely graphical representations.

Grammar-based formalisms ([Roads 85]) are also popular. Our suggested formalism is compatible with such approaches, because a parse tree can be expressed naturally in constituent terms. As we have seen above, different representational semantics are possible in this framework, depending on the interests of the user.

With respect to the dimensions of expressive completeness and structural generality, our framework allows more expressiveness than a traditional score, as the mixing of just and even tempered scales shows; it also allows the construction of musically significant structures over any or all the dimensions of the basic representation.

We have shown how our representation may be used for analysis and for composition in a musically intuitive way. We believe therefore that this framework provides an extensible vehicle for automated manipulation of descriptions of musical structures, and as such could allow musicians to gain access to the power of the computer in terms that make sense to the musician. However, this work will benefit in the future from more extensive worked examples, from usage by other musicians and musicologists and, eventually, from the development of software applications.

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.

[Boulez 85] P. Boulez. Quoi? quand? comment? In T. Machover, editor, *Quoi? Quand? Comment?*, pages 272–285, Paris, 1985. Christian Bourgois.

[Buxton & *et al* 78] W. Buxton *et al.* The use of hierarchy and instance in a data structure for computer music. *Computer Music Journal*, 2:10–20, 1978.

[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.

[Ebcioglu 88] K. Ebcioglu. An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12, 1988.

[Handel 89] S. Handel. *Listening.* MIT Press, Cambridge, MA, 1989.

[Jackendoff 87] R. Jackendoff. *Consciousness and the computational mind.* MIT Press, Cambridge, MA., 1987.

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

[Lewin 87] D Lewin. *Generalized Musical Intervals and Transformations.* Yale University Press, New Haven and London, 1987.

[Longuet-Higgins 62] H. C. Longuet-Higgins. Letter to a musical friend. *The musical review*, 23:244–8,271–80, 1962.

[Minsky 85] M. Minsky. Musique, sens et pensée. In T. Machover, editor, *Quoi? Quand? Comment?*, pages 137–63. Christian Bourgois, 1985.

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

[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.

[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. Also in *Computers and the Humanities*, vol. 27.

[Steedman 73] M.J. Steedman. *The Formal Description of Musical Perception*. Unpublished PhD thesis, Edinburgh University, 1973.

[Steedman 77] M.J. Steedman. The perception of musical rhythm and metre. *Perception*, 6:555–69, 1977.

[Whyte 91] J. Whyte. The automatic rhythmic analysis of monophonic music. AI/CS Undergraduate Project Report, Edinburgh University, 1991.

[Wiggins *et al* 89] G. Wiggins, M. Harris, and A. Smaill. Representing music for analysis and composition. In M. Balaban, K. Ebcioglu, O. Laske, C. Lischka, and L. Sorisio, editors, *Proceedings of the 2nd IJCAI AI/Music Workshop*, pages 63–71, Detroit, Michigan, 1989. Also available from Edinburgh as DAI Research Paper No. 504.

[Wolte 90] I. Wolte. Automatic music analysis. AI/CS Undergraduate Project Report, Edinburgh University, 1990.