Notes on CHARM - a Specification for the Representation of Musical Knowledge

Marcus T. Pearce

Centre for Cognition, Computation and Culture, Goldsmiths College,
University of London, New Cross, London SE14 6NW
m.pearce@gold.ac.uk

1 Introduction

The motivation for developing CHARM (Common Hierarchical Abstract Representation for Music) was to develop a logical specification of an abstract representation of music for use in a wide range of areas including composition, analysis and archiving. An abstract, logical scheme allows the flexible representation of many different kinds of musical structure at appropriate levels of generality for any particular task (Wiggins *et al.*, 1989). CHARM was intended to allow a general, expressive and flexible specification for the representation of music that would be independent of any particular style, tonal system, tradition or application (Smaill *et al.*, 1993).

2 Representing Events

The fundamental events in CHARM are discrete notes with constant pitch. CHARM was developed to describe music at the level of notes and phrases rather than the internal structure of individual tones and timbres (Harris *et al.*, 1991; Wiggins & Smaill, 2000), while the decision not to represent pitch in a continuous fashion is due to practical considerations (Wiggins *et al.*, 1993). Furthermore, CHARM is designed to represent performed musical objects (Wiggins & Smaill, 2000). This is because scores are viewed as performance instructions which may have several different interpretations (Harris *et al.*, 1991; Wiggins *et al.*, 1993). In consequence, features such as time and key signatures are not explicitly represented.

In order to describe a duration in terms of the interval between two points in time, Wiggins *et al.* (1989) define two sets *time* and *duration* containing possible onset/offset times and durations for events. They also define a function *dur(a,b)* which specifies the duration between times *a* and *b* and a distinguished duration denoted by 0 such that:

- *dur(x,y)* $= 0 \leftarrow x = y$;
- *dur(x,y)* + *dur(y,z)* = *dur(x,z)*;
- *dur(x,y)* = *-dur(x,y)*.

They also define a relation $\leq$ on the set *duration* and the operations + and - which allow us to compute the sum or difference of two members of *duration*. The set *duration* is therefore a linearly ordered commutative group.

An ordering $\sqsubseteq$ on *time* can be defined in terms of the ordering on duration as follows:

- $t_1 \sqsubseteq t_2 \leftrightarrow 0 \leq dur(t_1, t_2)$

Similarly, the functions $before_t$ and $after_t$ can be defined for all times *t* as bijections between time points and durations: $x \leftarrow dur(x,t)$ and $x \leftarrow dur(t,x)$ respectively.

Pitch is similarly represented by two sets *pitch* and *interval*. There is an ordering relation $\leq$ on *interval* and addition and subtraction functions. Finally, there is a measuring function *int* that gives the interval *int(a,b)* between any two members of *pitch*. Therefore, this formal scheme allows a uniform approach to representing time and pitch and similar approaches could be taken to the representation of dynamics and timbre.[1]

Any piece of music may then be represented by a set of event tuples describing each note in the piece:

```
event(Identifier,Pitch,Time,Duration,Amplitude,Timbre)
```

Each of the abstract data types (pitch, time etc.) will in addition have constructor and access functions associated with it. Harris *et al.* (1991) state that each data type *X* (pitch, time, duration, amplitude or timbre) must have an associated access function *getX* which returns the value of the type when passed any single event tuple. There must also be a function *putEvent* which returns the identifier associated with any event tuple passed to the function.

Smaill *et al.* (1993) note that the use of interface functions (e.g., *dur*, + etc.) allow the separation of implementation detail from the more important abstract structural information required for composition or analysis. They present an implementation in ML of the data type for pitch as a concrete example of the scheme. Finally, Harris *et al.* (1991) emphasise that the use of abstract data types facilitates the use of a common representation scheme across different concrete implementations of systems.

3 Representing Higher-order Structures

Harris *et al.* (1991) note that while it is widely recognised that musical representations must allow higher-level forms to be introduced hierarchically, they must not impose any one set of groupings on the user and should ideally allow the simultaneous assignment of different higher-level structures to any set of musical events. This allows both for different possible structural interpretations when there is ambiguity and the separation of distinct types of information about any set of events.

Constituents define higher-level groupings of events in CHARM and extend the framework to the description of hierarchical structure in music without committing the user to any particular hierarchy. The elements of a constituent are called particles and may be events or constituents but a constituent may not be a particle of itself (Harris *et al.*, 1991). As with events, each constituent must have appropriate typing and destructor functions associated with it (Harris *et al.*, 1991). A constituent is defined as the tuple:

```
constituent(Identifier,Properties,Definition,Particle_list,Description)
```[2]

where:

- the *identifier* is a unique identifier for the constituent;
- the *properties* or *structural type* defines the externally available properties of the constituent type which are derived from the externally defined interface functions;

[1] Wiggins & Smaill (2000) note that while it would be relatively straightforward to extend the internal structure of events to include such features of timbre, intensity and attack the existing implementation does not support them.

[2] Note that Wiggins *et al.* (1989) do not define the Properties components while neither they nor Smaill *et al.* (1993) define the Description component. Therefore, some of the examples in §4 will not feature these components of constituents.

- the *definition* or *musical type* defines the intrinsic musical properties of a constituent;
- the *particle list* contains a list of the particles of the constituent;
- the *description* is an arbitrary structure defined by the user for annotation of useful information.[3]

Structural types consist of a specification which is defined logically in terms of the comparison functions and their derivable relations and an environment list of the properties of the type. Smaill *et al.* (1993) give examples of two structural types:

- collection: defined by a start time and a duration but no further visible structure, e.g.,

```
constituent(co1,
            collection(0,34*crotchet),
            syrinx,
            [e000,e001,...e181,e182])
```

- stream: a strictly ordered collection of contiguous events defined by a start time and a duration (e.g., monophonic or homophonic music), e.g.,

```
constituent(co1,
            stream(0,34*crotchet),
            syrinx,
            [e000,e001,...e181,e182])
```

Smaill *et al.* (1993) note that a stream constituent could be used, for example, to call a version of a program optimised for monophonic lines. They also propose that any set of particles can be described by many different constituents with different musical or structural types and different identifiers. Harris *et al.* (1991) introduce the logical specification of a stream in terms of the interface functions getTime, getDuration and +. They also define the *slice* structural type whose particles all share a common time. Its properties therefore consist of that time and it is logically defined in terms of the accessor functions getTime, getDuration and +.

Smaill *et al.* (1993) argue that the small atomic set of interface functions they have defined for the pitch and time data types are sufficient to allow the construction of more complex operations for manipulating musical material in composition. Using the example of a motif in the form of a stream constituent,

```
constituent(c0,stream(0,t1),motif,[e1,e2,e3,e4,e5])
```

examples of such operations are:

- dilation of the interval structure between events: given a dilatation parameter, produce a constituent whenever the intervals of pitch are enlarged;
- dilation of the rhythmic structure between events: given a dilatation parameter, produce a constituent whenever the durations are enlarged;
- replacement of events by sub-constituents: events may be replaced by more complex constituents (e.g., chords, other motifs etc.);
- blanking out of material: remove elements in the constituent according to some criterion to break up an overly uniform structure;

[3] No interpretation is given to this component and while any software may write to it, no software using it in a strong sense can be guaranteed to be portable (Harris *et al.*, 1991).

- instrumentation: use distinct constituents for each voice;
- inversion of pitch and time: special cases of the dilatation operation.

An example of the use of CHARM in composition is provided by Harris *et al.* (1991).

4 Examples

Wiggins *et al.* (1989) present two examples of the use of CHARM for musical analysis. In the first, they use an algorithm due to Steedman (1977) which parses a sequence of durations into bars using a set of rules to extract rhythmic primitives (e.g., dactyls – long, short, short). Each note is represented as an event in which pitch is a triple of `[note,accidental,octave]`, start is the number of quavers from the start of the composition and duration is the length of an event in quaver notes. An example of an event is:

```
event(ev00,[g,natural,4],0,2,[]).
```

In the analysis, various constituents are constructed corresponding to important structural features, for example:

- a melody: `constituent(st15,melody,[ev00,ev04,ev24])`
- a dactyl grouping: `constituent(st02,dactyl,[ev03,ev05,ev07])`
- a single event group: `constituent(st15,single,[ev00])`
- a metric chunk: `constituent(st50,metric_chunk,[st03,st04])`
- an ordering on the metric chunks: `constituent(st54,metric_ordering,[st50,st51,st52,st53])`

Note that the `metric_chunk` and `metric_ordering` constituents have other constituents making up their particle list. Harris *et al.* (1991) introduce a formal definition of the structural type dactyl in terms of the interface functions for the basic data types.

In a second example, Wiggins *et al.* (1989) use CHARM for an analysis of *Syrinx* by Debussy using an algorithm due to Ruwet (1972) and used by Nattiez (1975) for the same purpose. Here we focus on the more complex analysis presented by Smaill *et al.* (1993) which illustrates the use of the structural and musical types of constituents. Smaill *et al.* (1993) define four types of similarity used in the analysis:

- identity: a phrase is identical to another;
- longer identity: identity except for first note which is tied to a preceding note;
- transposition: a phrase is a transposition by constant number of semitones of an existing motif;
- loose transpose: a phrase is a transposition of an existing motif regardless of the equality of note durations.

Smaill *et al.* (1993) describe the algorithm as follows. First, the piece is scanned for phrases which are repeats of an earlier phrase under the identity relation. When such a repeat is found, the first occurrence is called a *motif* and the repeats *derivations* under the *identity* similarity. All derivations are removed from the search space. Second, the piece is scanned for phrases which are similar to the motif under the other similarity metrics. These are given special status

as derivations under the other similarity relations. These steps are repeated until no further repeated phrases can be found.[4]

Since the musical groupings found by the program are all strictly ordered contiguous sequences of events, they may be represented as stream constituents. An example of the output is as follows:

```
constituent(co1,stream(0,dotted_minim),motif(mtf2),
            [e000,e001,e002,e003,e004,e005,e006,e007])
constituent(co2,stream(3*dotted_minim),derived(identity,mtf2),
           [e014,e015,e016,e017,e018,e019,e020,e021,e022,e023])
```

which states that constituent `co2` is an identical derivation of the motif `co1`. This example makes clear the distinction between the structural type of a constituent and the musical type which is used here to declare the special status of the motif and the derived phrase.

In order to demonstrate the expressive power of CHARM, Wiggins *et al.* (1989) represented duration as `Bar_number+Beat_number` and obtained exactly the same results on the rhythmic analysis task described above. In a similar vein, Smaill *et al.* (1993) obtained exactly the same results in the analysis of Syrinx when pitch (intervals) were represented as integer numbers of quartertones and onset times and durations were represented as rational fractions of demisemiquavers. Furthermore, they use this representation to analyse the piece *Largo* from the *Three Quartertone Pieces for Two Pianos* by Debussy which is written in a different tonal system from *Syrinx*. Because the piece is non-monophonic, it was necessary to search the collection constituent for stream constituents online. Although, the program did arrive at a limited structural interpretation of the piece, Smaill *et al.* (1993) suggest that an extension to this approach would define a new `chord` constituent and then look for streams of chords.

5 Advantages

Wiggins *et al.* (1993) introduce two orthogonal dimensions along which representation systems for music may be classified: *expressive completeness* and *structural generality*. The former refers to the range of raw musical data that can be represented while the latter refers to the range of high-level structures that may be represented and manipulated. For example, waveforms have high completeness but low generality while traditional scores have high generality but restricted expressive completeness. Different tasks will place different emphasis on each of the properties; archiving, for example, places a stress on accuracy of storage and recall and requires high expressive completeness while for analysis and composition, structural generality is more important.

Wiggins *et al.* (1993) argue that the CHARM system scores well on both expressive completeness and structural generality. Regarding the latter, the use of constituents allows the explicit representation of any structural property of music at any level of abstraction and a precise characterisation of the relationships between such structures (for example, multiple viewpoints of any musical structure may be easily defined). Furthermore, the use of abstract data types facilitates the construction of functions for manipulating these musical structures. In terms of expressive completeness, the abstraction away from implementational details resulting from the use of abstract data types frees the system from any particular style, tonal system, tradition or

[4]Note that in contrast to the algorithm used by Ruwet (1972), this algorithm requires that a phrase be exactly repeated before it is considered under the non-identity similarities. This has the effect of considerably reducing the search space.

application (see §3) as long as the mathematical specifications are followed. Finally, this abstraction from detail also facilitates the common use of a general and expressive representation scheme for many different applications.

References

Harris, M., Smaill, A., & Wiggins, G. (1991). Representing music symbolically. In A. Camurri & C. Canepa (Eds.), *IX Colloquio di Informatica Musicale*. Genova, Italy: Universita di Genova.

Nattiez, J. J. (1975). *Fondements d'une sémiologie de la musique*. Paris: Union Générale d'Editions.

Ruwet, N. (1972). *Langage, Musique et poésie*. Paris: Editions du Seuil.

Smaill, A., Wiggins, G. A., & Harris, M. (1993). Hierarchical music representation for composition and analysis. *Computers and the Humanities*, *27*, 7–17.

Steedman, M. J. (1977). The perception of musical rhythm and metre. *Perception*, *6*(5), 555–569.

Wiggins, G. A., Harris, M., & Smaill, A. (1989). Representing music for analysis and composition. In M. Balaban, K. Ebcioğlu, O. Laske, C. Lischka, & L. Soriso (Eds.), *Proceedings of the Second Workshop on AI and Music* (pp. 63–71). Menlo Park, CA: AAAI.

Wiggins, G. A., Miranda, E., Smaill, A., & Harris, M. (1993). A framework for the evaluation of music representation systems. *Computer Music Journal*, *17*(3), 31–42.

Wiggins, G. A. & Smaill, A. (2000). What can artificial intelligence bring to the musician? In E. R. Miranda (Ed.), *Readings in Music and Artificial Intelligence* (pp. 29–46). Amsterdam: Harwood Academic Publishers.