SCORES LEVEL COMPOSITION BASED ON THE GUIDO MUSIC NOTATION

Dominique Fober, Yann Orlarey, Stéphane Letz

SCORES LEVEL COMPOSITION BASED ON THE GUIDO MUSIC NOTATION

D. Fober, Y. Orlarey, S. Letz

Grame
Centre national de crétaion musicale, Lyon, France
fober@grame.fr

ABSTRACT

Based on the Guido Music Notation format, we have developed tools for music score *"composition"* (in the etymological sense), i.e. operators that take scores both as target and arguments of high level transformations, applicable for example to the time domain (e.g. cutting the head or the tail of a score) or to the structural domains (e.g. putting scores in sequence or in parallel). Providing these operations at score level is particularly convenient to express music ideas and to compose these ideas in an homogeneous representation space. However, scores level composition gives raise to a set of issues related to the music notation consistency. This paper introduces the GUIDO Music Notation format, presents the score composition operations, the notation issues and a proposal to solve them.

1. INTRODUCTION

The GUIDO Music Notation format [GMN] [4] has been designed by H. Hoos and K. Hamel more than ten years ago. It is a general purpose formal language for representing score level music in a platform independent plain text and human readable way. It is based on a conceptually simple but powerful formalism: its design concentrates on general musical concepts (as opposed to graphical characteristics). A key feature of the GUIDO design is adequacy which means that simple musical concepts are represented in a simple way and only complex notions require complex representations.

Based on the GMN language, the GUIDO Library [2, 3] provides a powerful score layout engine that differentiates from the compiler solutions for music notation [5, 1] by its ability to be embedded into standalone applications, and by its fast and efficient rendering engine, making the system usable in *real-time* for simple music scores.

Based on the combination of the GUIDO language and engine, score level composition operators have been designed, providing time or pitch transformations, composition in sequence or in parallel, etc. Developing score level composition operators provides an homogeneous way to write scores and to manipulate them while remaining at a high music description level. Moreover, the design allows to use scores both as target and as arguments of the operations, enforcing the notation level metaphor.

However, applied at score level, these operations raise a set of issues related to the music notation consistency. We propose a simple typology of the music notation elements and a set of rules based on this typology to enforce the music notation coherence.

The next section introduces the GUIDO Music Notation format, followed by a presentation of the score composition operations, the related notation problems and the proposed solutions, including a language extension to handle reversibility issues.

2. THE GUIDO MUSIC NOTATION FORMAT

2.1. Basic concepts

Basic GUIDO notation covers the representation of notes, rests, accidentals, single and multi-voiced music and the most common concepts from conventional music notation such as clefs, meter, key, slurs, ties, beaming, stem directions, etc. Notes are specified by their name `(a b c d e f g h)`, optional accidentals ('#' and '&' for sharp and flat), an optional octave number and an optional duration. Duration is specified in one of the forms:

```
'*'enum'/'denom dotting
'*'enum dotting
'/'denom dotting
```

where *enum* and *denom* are positive integers and *dotting* is either empty, '.', or '..'. When *enum* or *denom* is omitted, it is assumed to be 1. The duration represents a whole note fractional.

When omitted, optional note description parts are assumed to be equal to the previous specification before in the current sequence.

Chords are described using comma separated notes enclosed in brackets e.g `{c, e, g}`

2.2. GUIDO tags

Tags are used to represent additional musical information, such as slurs, clefs, keys, etc. A basic tag has one of the forms:

```
\tagname
\tagname<param-list>
```

where `param-list` is a list of string or numerical arguments, separated by commas (','). In addition, a tag may have a time range and be applied to a series of notes (e.g. slurs, ties, etc.); the corresponding forms are:

```
\tagname(note-series)
\tagname<param-list>(note-series)
```

The following GMN code illustrates the concision of the notation; figure 1 represents the corresponding GUIDO engine output.

```
[ \meter<"4/4"> \key<-2> c d e& f/8 g ]
```

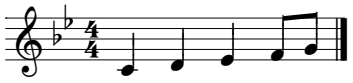

Figure 1. A simple GMN example

2.3. Notes sequences and segments

A note sequence is of the form `[tagged-notes]` where `tagged-notes` is a series of notes, tags, and tagged ranges separated by spaces. Note sequences represent single-voiced scores. Note segments represent multi-voiced scores; they are denoted by `{seq-list}` where `seq-list` is a list of note sequences separated by commas as shown by the example below (figure 2):

```
{ [ e g f ], [ a e a ] }
```

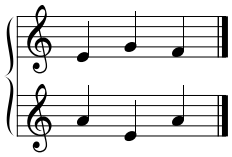

Figure 2. A multi-voices example

2.4. Advanced GUIDO.

The advanced GUIDO specification extends basic GUIDO with more tags and more tags parameters, giving more control over the score layout. For example, it introduces tags parameters like *dx* and *dy* for fine positioning of the score elements, notes and rests format specifications, staff assignments, etc.

3. COMPOSING MUSIC SCORES

Since GUIDO is a concise textual format, it seems natural to use operations commonly applied to text, like cut, copy and paste, text concatenation, etc. Thus the first idea with the score level operations was based on textual manipulation, extended to music specific operations.

3.1. Operations

Score level operations are given by table 1. These operations are available as library API calls, as command line tools, or using a graphic environment named GUIDOCalculus. Almost all of the operations take a GMN score and a value parameter as input and produce a GMN score as output. The value parameter can be taken from another GMN score: for example, the `top` operation cuts the bottom voices of a score after a given voice number; when using a score as parameter, the voice number is taken from the score voices count.

All the operations concentrate on the transformed dimension (pitch, time), without modifying user defined elements or trying to interfere with the automatic layout of the GUIDO Engine (that may add notation elements like clef, barlines). For example, the *duration* operation recomputes the notes length but doesn't affect the time signature or the barlines. When two scores are put in parallel, the system preserves each voice time and key signatures, even when they don't match. The transposition operation is the only exception: it adds or modifies the key signature and selects the simplest enharmonic diatonic transposition.

The design allows all the operations to take place consistently at the notation level. Using the command line tools, series of transformations can be expressed as pipelining scores through operators e.g.

```
head s1 s2 | par s2 | transpose "[ c ]"
```

3.2. Notation issues

Actually, the score level composition functions operate on a memory representation of the music notation. But we'll illustrate the notation issues with the textual representation which is equivalent to the memory representation.

Let's take an example with the `tail` operation applied to the following simple score:

```
[\clef<"f"> c d e c]
```

A raw cut of the score after 2 notes would give `[e c]`, removing the clef information and potentially leading to unexpected results (figure 3).

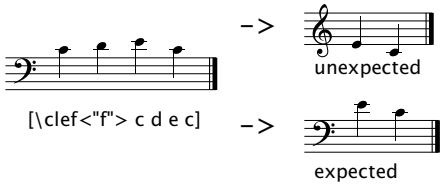

Figure 3. Tail operation consistency

Here is another example with the `seq` operation: a raw sequence of `[\clef<"g"> c d]` and `[\clef<"g"> e c]` would give `[\clef<"g"> c d \clef<"g"> e c ]` where the clef repetition (figure 4) is useless and blurs the reading.

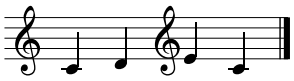

Figure 4. A raw sequence operation

Some operations may also result in syntactically incorrect results. Consider the following code:

```
[g \slur(f e) c]
```

operation	args	description
seq	$s1$ $s2$	puts the scores $s1$ and $s2$ in sequence
par	$s1$ $s2$	puts the scores $s1$ and $s2$ in parallel
rpar	$s1$ $s2$	puts the scores $s1$ and $s2$ in parallel, right aligned
top	$s1$ [n | $s2$]	takes the n top voices of $s1$; when using a score $s2$ as parameter, n is taken from $s2$ voices count
bottom	$s1$ [n | $s2$]	takes the bottom voices of $s1$ after the voice n; when using a score $s2$ as parameter, n is taken from $s2$ voices count
head	$s1$ [d | $s2$]	takes the head of $s1$ up to the date d; when using a score $s2$ as parameter, d is taken from $s2$ duration
evhead	$s1$ [n | $s2$]	id. but on events basis i.e. the cut point is specified in n events count; when using a score $s2$ as parameter, n is taken from $s2$ events count
tail	$s1$ [d | $s2$]	takes the tail of a score after the date d; when using a score $s2$ as parameter, d is taken from $s2$ duration
evtail	$s1$ [n | $s2$]	id. but on events basis i.e. the cut point is specified in n events count; when using a score $s2$ as parameter, n is taken from $s2$ events count
transpose	$s1$ [i | $s2$]	transposes $s1$ to an interval i; when using a score $s2$ as parameter, i is computed as the difference between the first voice, first notes of $s1$ and $s2$
duration	$s1$ [d | r | $s2$]	stretches $s1$ to a duration d or using a ratio r; when using a score $s2$ as parameter, d is computed from $s2$ duration
applypitch	$s1$ $s2$	applies the pitches of $s1$ to $s2$ in a loop
applyrythm	$s1$ $s2$	applies the rhythm of $s1$ to $s2$ in a loop

Table 1. Score level operations

slicing the score in 2 parts after `f` would result in

a) `[g \slur(f]` and **b)** `[e) c]`

i.e. with uncompleted range tags. We'll use the terms *opened-end* tags to refer the a) form and *opened-begin* tags for the b) form.

These simple examples illustrate the problem and there are many more cases where the music notation consistency has to be preserved across score level operations.

4. MUSIC NOTATION CONSISTENCY

In order to solve the notation issues, we propose a simple typology of the notation elements regarding their time extent and a set of rules defining adequate consistency policies according to the operations and the elements type.

4.1. Notation elements time extent

The GMN format makes a distinction between *position* tags (e.g. `\clef`, `\meter`) and *range* tags (e.g. `\slur`, `\beam`). Position tags are simple notations marks at a given time position while range tags have an explicit time extent: the duration of the enclosed notes. However, this distinction is not sufficient to cover the time status of the elements: many of the position tags have an implicit time duration and generally, they last up to the next similar notation or to the end of the score. For example, a dynamic lasts to the next dynamic or the end of the score.

Table 2 presents a simple typology of the music notation elements, mainly grounded on their time extent. Based on this typology, provisions have to be made when:

- computing the beginning of a score:
  - **1)** the pending explicit time extent elements must be properly opened (i.e. *opened-begin* tags, see section 3.2)
  - **2)** the current implicit time extent elements must be recalled,
- computing the end of a score:
  - **3)** the explicit time extent elements must be properly closed (i.e. *opened-end* tags)
- putting scores in sequence:
  - **4)** implicit time extent elements starting the second score must be skipped when they correspond to current existing elements.

4.2. Structure control issues

Elements relevant to the *others / structure control* time extent category may also give rise to inconsistent notation: a *repeat begin* bar without *repeat end*, a *dal segno* without *segno*, a *da capo al fine* without *fine*, etc. We introduce new rules to catch the repeat bar issue. Let's first define a *pending* repeat end as the case of a voice with a repeat begin tag without matching repeat end.

5) when computing the end of a score, every *pending* repeat end must be closed with a repeat end tag.

6) from successive unmatched repeat begin tags, only the first one must be retained.

time extent	description	sample
explicit	duration is explicit from the notation	slurs, cresc.
implicit	element lasts to the next similar element or to the end of the score	meter, dynamics, key
others	structure control	coda, da capo, repeats
-	formatting instructions	new line, new page
-	misc. notations	breath mark, bar

Table 2. Typology of notation elements.

7) from successive repeat end tags, only the last one must be retained.

No additional provision is made for the other structure control elements: possible inconsistencies are ignored but this choice preserves the operations reversibility.

4.3. Operations reversibility

The above rules solve most of the notation issues but they do not permit the operations to be reverted: consider a score including a slur, sliced in the middle of the slur and reverted by putting the parts back in sequence. The result will include two slurs (figure 5) due to the rules **1)** and **3)** that enforce opening *opened-begin* tags and closing *opened-end* tags.

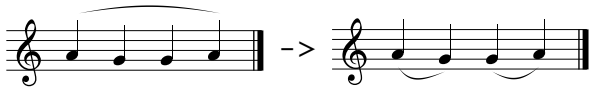

Figure 5. A score sliced and put back in sequence

To solve the problem, we need the support of the GMN language and we introduce a new tag parameter, intended to keep the history of range tags and to denote *opened-end* and/or *opened-begin* ancestors. The parameter has the form:

```
open="type"
```

where `type` is in [`begin`, `end`, `begin-end`], corresponding to *opened-begin*, *opened-end*, and *opened-begin-end* ancestors.

Next, we introduce a new rule for score level operations. Let's first define *adjacent* tags as tags placed on the same voice and that are not separated by any note or chord. Note that range tags are viewed as containers and thus, notes in the range do not separate tags.

8) *adjacent* similar tags carrying an `open` parameter are mutually cancelled when the first one is *opened-end* and the second one *opened-begin*.

For example, the application of this rule to the following score:

```
[ \anytag<open="end">(f g)
\anytag<open="begin">(f e) ]
```

will give the score below:

```
[ \anytag(f g f e) ]
```

Note that Advanced GUIDO allows range tags to be expressed using a `Begin` and `End` format (e.g. `\slurBegin`, `\slurEnd` instead `\slur(range)`). This format is handled similarly to regular range tags and the `open` parameter is also implemented for Begin/End tags.

5. CONCLUSION

Music notation is complex due to the large number of notation elements and to the heterogeneous status of these elements. The typology proposed in table 2 is actually a simplification intended to cover the needs of score level operations but it is not representative of this complexity. However, it reflects the music notation semantic and could be reused with other score level music representation language. Thus apart for the reversibility rule that requires the support of the music representation language, all the other rules are independent from the GMN format and applicable in other contexts.

Score level operations could be very useful in the context of batch processing (e.g. voices separation from a conductor, excerpt extraction, etc.). The operations presented in table 1 support this kind of processing but they also open the door to a new approach of the music creative process.

6. REFERENCES

[1] A. E. Daniel Taupin, Ross Mitchell. Musixtex using tex to write polyphonic or instrumental music. [Online]. Available: http://icking-music-archive.org/

[2] C. Daudin, D. Fober, S. Letz, and Y. Orlarey, "The Guido Engine - a toolbox for music scores rendering." in *Proceedings of the Linux Audio Conference 2009*, 2009, pp. 105–111.

[3] D. Fober, S. Letz, and Y. Orlarey, "Open source tools for music representation and notation." in *Proceedings of the first Sound and Music Computing conference - SMC'04*. IRCAM, 2004, pp. 91–95.

[4] H. Hoos, K. Hamel, K. Renz, and J. Kilian, "The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music." in *Proceedings of the International Computer Music Conference*. ICMA, 1998, pp. 451–454.

[5] H.-W. Nienhuys and J. Nieuwenhuizen, "LilyPond, a system for automated music engraving." in *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, May 2003.