

SCORES COMPOSITION BASED ON THE GUIDO MUSIC NOTATION

ABSTRACT

Based on the Guido Music Notation format, we have developed tools for music score composition, i.e. operators taking score both as target and as 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 the score level is particularly convenient to express music ideas and to compose these ideas in an homogeneous representation space. However, scores composition gives raise to a set of issues related to the music notation consistency. The 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 features). 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 composition operators have been designed, providing rhythmic or pitch transformations, time selections, composition in sequence or in parallel, etc.

Music notation is the most common representation used by musicians. 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.

As a solution, 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.

This paper introduces the GUIDO Music Notation format, the next sections present the score composition operations, the related notation issues and the proposed solutions.

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 ] }
```

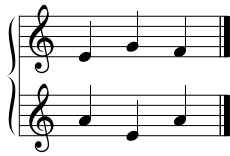


Figure 2. A multi-voices example

2.4. Advanced GUIDO.

The advanced GUIDO specification provides more tags and more control over the score layout. In particular, it introduces tags parameters like `dx` and `dy` for fine positioning of the score elements, notes and rests format specifications, staff assignments, etc. Below is an example of advanced guido with the corresponding output (figure 3).

```
{
  [
    \barFormat<"system">
    \staff<1> \stemsUp \meter<"2/4">
    \intens<"p", dx=1hs,dy=-7hs>
    \beam(g2/32 e/16 c*3/32) c/8
    \beam(\noteFormat<dx=-0.9hs>(a1/16) c2 f)
    \beam(g/32 d/16 h1*3/32) d2/8
    \beam(h1/16 d2 g)],
    [\staff<1>\stemsDown g1/8 e
     f/16 \noteFormat<dx=0.8hs>(g) f a a/8 e
     f/16 g f e],
    [\staff<2> \meter<"2/4">
     \stemsUp a0 f h c1],
    [\staff<2> \stemsDown c0 d g {d, a}]
  ]
}
```

3. COMPOSING MUSIC SCORES

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 as well. Almost all of the operations take a GMN score and a value parameter as input and produce

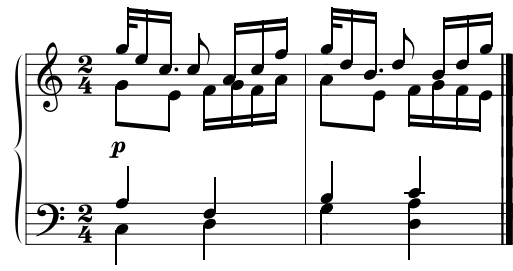


Figure 3. An advanced Guido example

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.

This design allows all the operations to take place consistently at the notation level. In addition and 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 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]` as result, removing the clef information and potentially leading to unexpected results (figure 4).

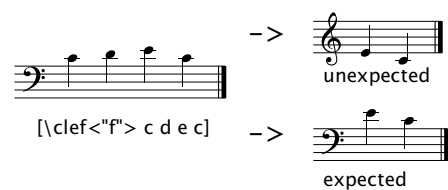


Figure 4. Tail operation consistency

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 5) is useless and blurs the reading.



Figure 5. A raw sequence operation

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
applyrhythm	$s1\ s2$	applies the rhythm of $s1$ to $s2$ in a loop

Table 1. Score level operations

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

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

slicing the score in 2 parts after f would result in

a) $[g \ \backslash\text{slur}(f)]$ 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 but there are many more cases where the music notation consistency has to be preserved across score level operations.

4. MUSIC NOTATION CONSISTENCY

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. $\backslash\text{clef}$, $\backslash\text{meter}$) and *range* tags (e.g. $\backslash\text{slur}$, $\backslash\text{beam}$). 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 problem: 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 dynamics last to the next dynamic or the end of the score.

The 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 (i.e. *opened-begin* tags, see section 3.2) must be properly opened
 - 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.

Concerning the *other* time extent category, provisions are made for the *structure control* elements (see section 4.3).

4.2. 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 6) due to the rules 1)

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
-	notation marks	breath mark, bar

Table 2. Typology of notation elements.

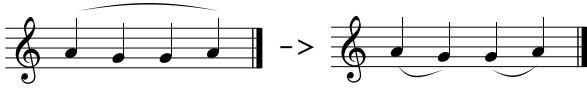


Figure 6. A score sliced and put back in sequence

and 3) that enforce opening *opened-begin* tags and closing *opened-end* tags.

To solve the problem, we need the support of the GMN language: we introduce a new tag parameter, keeping the history of a range tag, and indicating when it corresponds to an *opened-end* and/or *opened-begin* tag. The parameter has the form:

`\tag<open="type">`

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

Next, we introduce a new rule for score level operations. Let's first define *adjacent* tags as tags placed on the same voice and not separated by any note or chord.

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

Thus when a score level operation encounters a form like:

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

it should transform it to:

`\anytag(f g f e)`

4.3. 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.

- when computing the end of a score, every *pending* repeat end must be closed with a repeat end tag.
- from successive unmatched repeat begin tags, only the first one must be retained.
- 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.

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.

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.