SCORES COMPOSITION BASED ON THE GUIDO MUSIC NOTATION

D. Fober, C. Daudin, S. Letz, Y. Orlarey
Grame - Centre national de création musicale
{fober, daudin, letz, orlarey}@grame.fr

ABSTRACT

Based on the Guido Music Notation format, we have developed a library - the GuidoAR library - that provides a simple and efficient memory representation of the music notation as well as score level operations. Applying operations at score level (like cutting the score head or tail) gives raise to a set of issues related to the music notation consistency. We present the Guido Music Notation format, the GuidoAR library, the score composition operations, the music notation related issues and a proposed way to solve them.

1 INTRODUCTION

The Guido Music Notation format (GMN) [1] [2] has been designed by H. Hoos and K. Hamel more than ten years ago. It is very close to the Lilypond format[3] [4] but it has appeared before. The GMN format 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.

Computer music is rich of many score level music representation languages [5][6][7][8][9]. This paper doesn't aim at making a comparison between these languages and the main reason we chose the GMN format for this work is the language simplicity and readability, in combination with the availability of the Guido Engine [10][11], a powerful open source C/C++ library for music score layout and graphic rendering based on the GMN format, providing an easy way to dynamically experiment with score level operations.

Based on the GMN format, we have developed a C++ library, the GuidoAR library, more suitable to implement score composition operations than the Guido Engine internal memory representation. Composition operations are basic operations like transposition, cutting the head or the tail of a score, putting scores in sequence or in parallel, etc. Implementing these operations gave quickly raise to a set of issues related to the music notation consistency. To solve these issues, we propose a simple typology of the music notation elements and a set of rules based on this typology to enforce the music notation consistency.

Music notation is the most common representation used by musicians. Developing score level composition operations 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.

This paper introduces first the Guido Music Notation format, next the GuidoAR library is presented and the last section presents the score composition operations, the related notation issues and the proposed solution.

SMC 2009, July 23-25, Porto, Portugal

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 '..', with the same semantic than the music notation. 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 (like slurs, ties etc.); the corresponding form is:

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

The following GMN code illustrates the concision of the notation; figure 2 represents the corresponding Guido engine output.

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

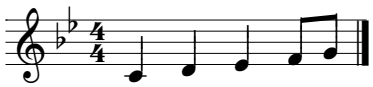

Figure 1. A simple GMN example

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:

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

The corresponding output is given by figure 2.

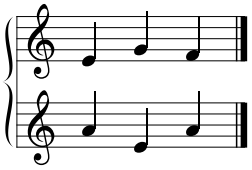

Figure 2. A multi-voices example

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 to indicate exact 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
  \beam(f/16 \noteFormat<dx=0.8hs>(g) f a)
   a/8 e
  \beam(f/16 g f e)],

 [\staff<2> \meter<"2/4">
  \stemsUp a0 f h c1],
 [\staff<2> \stemsDown c0 d g {d, a}]
}
```

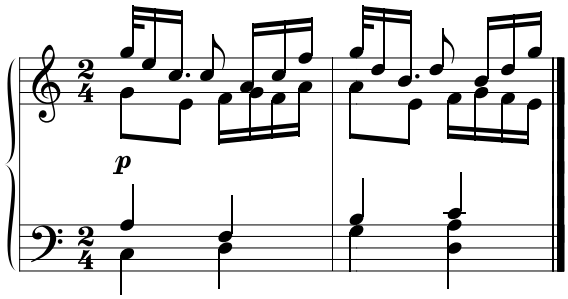

Figure 3. An advanced Guido example

3 THE GUIDOAR LIBRARY

The GuidoAR library is a C/C++ framework providing a memory representation of the Guido Music Notation [GMN] as well as basic operations on this representation. The library name - GuidoAR - stands for Guido Abstract Representation, referring to the GuidoEngine [12] [10] terminology and opposed to the Guido Graphic Representation : the abstract representation is intended as the pending memory representation of the GMN and deals with the logical score layout (stems direction, beaming, etc.) while a graphic representation is intended to provide exact graphic formatting information of the score.

The main issues in designing a C++ library to support the GMN format are related to the significant number of notation elements: in addition to notes, chords, voices, the format includes over hundred tags. Thus the cost of describing all the notations elements, the design of an adequate and efficient memory representation and the easiness to maintain and to manipulate have been at the center of the library design.

3.1 Notation elements representation

The GuidoAR library is based on a single `guidoelement` class that contains a name and an attributes list (figure 4). A `guidoattribute` has a name, a value and an optional unit (used for exact positioning).

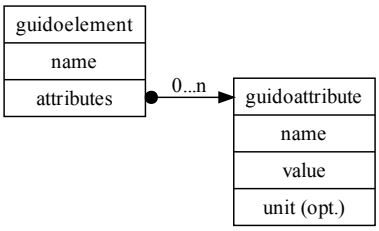

Figure 4. Basic objects.

The `guidoelement` class is specialized to represent the main elements of the notation (figure 5). Guido tags are consistently covered by the `guidotag` class, derived into as many types as existing tags using templates.

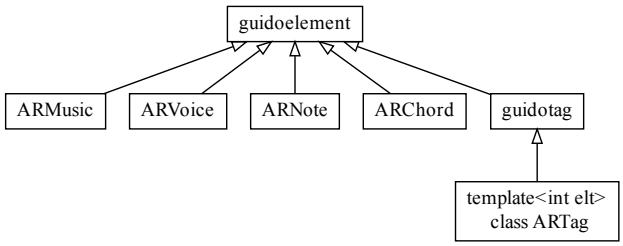

Figure 5. The notation elements classes

An automatic memory management using smart pointers simplifies the programming task. This homogeneous design leads to simplicity.

3.2 Browsing the representation

The memory representation is organized into a tree similarly to the GMN hierarchy. The elements support the acyclic visitor pattern [13] and STL iterators as well, providing simple and powerful ways to browse the music representation. The following sample code illustrates how to count the notes of a score using a visitor design:

```
struct countnotes : public visitor<SARNote> {
  int fCount;
        countnotes() : fCount(0) {}
  void visitStart(SARNote&) {fCount++;}
};
```

And the following code implements the same feature with STL iterators and the `count_if` algorithm.

```
struct countnotespredicat {
  bool operator () (const Sguidoelement elt)
  { return
   (dynamic_cast<ARNote*>((guidoelement*)elt)
    ? true : false); }
};
countnotespredicat p;
count = count_if(score->begin(),score->end(),p);
```

These mechanisms are at the basis of all the score composition operations supported by the library.

4 SCORES COMPOSITION

4.1 Basic operations

Score level operations provided by the GuidoAR library are given by the table 1. These operations are available as library API calls and as command line tools as well. Almost all of the operations take a GMN score and a value parameter as input and produce a GMN score as output, but the value parameter can also be indicated using another GMN score. For example, the `guidotop` 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.

4.2 Notation issues

Actually and to preserve the notation consistency, most of the score level operations are more complex than simply cutting or adding a branch in the memory representation. We'll illustrate the problem with the textual representation which is equivalent to the memory representation. Let's take the example of the `tail` operation with the following simple score:

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

A raw cut of the score after 2 notes would give the `[ e c]` result which would be naturally rendered as illustrated by the figure 6 while the use of the same clef is expected.

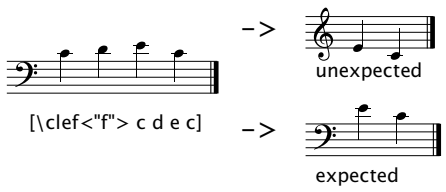

Figure 6. Tail operation consistency

Another example with the `sequence` operation: a raw sequence of `[\clef<"g"> c d]` and `[\clef<"g"> e c]` would give `[\clef<"g"> c d \clef<"g"> e c ]` as result (figure 7) while the clef repetition is unexpected.

Table 1. Score level operations

tool name and args	description
guidoseq $s1$ $s2$	puts the scores $s1$ and $s2$ in sequence
guidopar $s1$ $s2$	puts the scores $s1$ and $s2$ in parallel
guidorpar $s1$ $s2$	puts the scores $s1$ and $s2$ in parallel but right aligned
guidotop $s1$ $[n \mid s2]$	takes the n top voices of $s1$; when using a score $s2$ as parameter, n is taken from $s2$ voices count
guidobottom $s1$ $[n \mid s2]$	takes the bottom voices of $s1$ after the n voice; when using a score $s2$ as parameter, n is taken from $s2$ voices count
guidohead $s1$ $[d \mid s2]$	takes the head of $s1$ up to the date d; when using a score $s2$ as parameter, d is taken from $s2$ duration
guidoevhead $s1$ $[n \mid s2]$	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
guidotail $s1$ $[d \mid s2]$	takes the tail of a score after the date d; when using a score $s2$ as parameter, d is taken from $s2$ duration
guidoevtail $s1$ $[n \mid s2]$	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
guidotranspose $s1$ $[i \mid s2]$	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 $s2$
guidoduration $s1$ $[d \mid f \mid s2]$	stretches $s1$ to a duration d or using a factor f; when using a score $s2$ as parameter, d is computed from $s2$ duration
guidoapplypitch $s1$ $s2$	applies the pitches of $s1$ to $s2$; the pitches list is applied in a loop up to the end of $s2$
guidoapplyrythm $s1$ $s2$	applies the rhythm of $s1$ to $s2$; the durations list is applied in a loop up to the end of $s2$

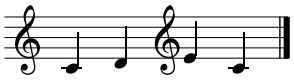

Figure 7. A raw sequence operation

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

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

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

There are many more cases, where the music notation consistency has to be preserved through scores composition operations.

4.3 Notation elements time extend

One way to solve the problem is to make a typology of the notation elements regarding their time extend and to define adequate consistency policies according to the operations and the element type.

The GMN format makes a distinction between *position* tags (like `\clef` or `\meter`) and *range* tags (`\slur`, `\beam`, etc.): range tags have an explicit time extend (the duration of the enclosed notes) and position tags are simply notations marks at a given time position. However, this distinction is not sufficient to cover the problem: many of the position tags have an implicit time duration and they generally last up to the next similar notation or to the end of the score. For example, a `\meter` tag lasts to the end of the score or to the next meter notation. The table 2 presents a simple typology

Table 2. Typology of notation elements.

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

of the music notation elements. All the Guido range tags correspond to the explicit time extend elements and most of the position tags have an implicit time extend. Based on this typology, a set of simple rules for score level operations could partially solve the notation issues:

- when computing the end of a score, all the explicit time extend elements must be properly closed,
- when computing the beginning of a score, all the pending explicit time extend elements (i.e. *opened-begin* tags) must be recalled,
- when computing the beginning of a score, all the current implicit time extend elements must be recalled,
- when putting scores in sequence, implicit time extend elements starting the second score must be skipped when they correspond to current existing elements.

No provision is made for the notation elements falling into the *other* time extend category except for the *structure control* elements (see section 4.5).

4.4 Operations reversibility

The above rules solve most of the notation consistency issues but they do not permit the operations to be reversible. Consider a score including a slur, sliced in the middle of this slur and restored by putting the parts back in sequence. The resulting score will include two slurs (figure 8) instead one, due to the rules that enforce closing *opened-end* tags and starting *opened-begin* tags.

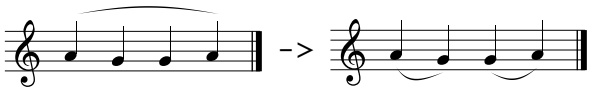

Figure 8. A score sliced and put back in sequence

Here to solve the problem, we'll need the support of the GMN language. The idea is to keep the history of the range tags by introducing a new tag parameter to indicate tags creation corresponding to *opened-end* and *opened-begin* tags. The parameter has the form:

`\tag<open="type">`
where `open` is the parameter name
and `type` is between `begin` and `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:

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

it should transform it to:

`\tag(f g f e)`

which solves the reversibility issue.

4.5 Structure control issues

Elements relevant to the *others / structure control* time extend 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 2 new rules to catch the repeat bars 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 and 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 proposed typology in table 2 is actually an arbitrary simplification intended to cover the needs of score level operations but is not representative of this complexity. However, it is based on the music notation semantic and thus could be reused with any score level music representation language. Apart from the reversibility rules defined in 4.4 that require the support of the music representation language to operate, all the other rules are independent from the GMN format. Most of the elements relevant to the *others* time extend category have been ignored by the proposed rules, which may result in *non-standard* layout (e.g. incoherent formatting instructions), but it preserves the operations reversibility. Thus the score composition operations do not guaranty a correct automatic music layout. In a future work, we plan to transform the notation into a lambda calculus based programming language [14] with these operations as language operators.

6 REFERENCES

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

[2] H. H. Hoos and K. A. Hamel. The GUIDO Music Notation Format Specification - version 1.0, part 1: Basic GUIDO. Technical report TI 20/97, Technische Universitat Darmstadt, 1997.

[3] Han-Wen Nienhuys and Jan Nieuwenhuizen. LilyPond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, May 2003.

[4] Han-Wen Nienhuys. Lilypond, automated music formatting and the art of shipping. In *Forum Internacional Software Livre 2006 (FISL7.0)*, 2006.

[5] Walter B. Hewlett. MuseData: Multipurpose Representation. In Selfridge-Field E., editor, *Beyond MIDI, The handbook of Musical Codes.*, pages 402–447. MIT Press, 1997.

[6] David Huron. Humdrum and Kern: Selective Feature Encoding. In Selfridge-Field E., editor, *Beyond MIDI, The handbook of Musical Codes.*, pages 376–401. MIT Press, 1997.

[7] E. Selfridge-Field. DARMS, Its Dialiects, and Its Uses. In *Beyond MIDI, The handbook of Musical Codes.*, pages 163–174. MIT Press, 1997.

[8] Smith Leland. SCORE. In *Beyond MIDI, The handbook of Musical Codes.*, pages 252–280. MIT Press, 1997.

[9] M. Good. MusicXML for Notation and Analysis. In W. B. Hewlett and E. Selfridge-Field, editors, *The Virtual Score*, pages 113–124. MIT Press, 2001.

[10] 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*, pages 91–95. IRCAM, 2004.

[11] 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*, pages 105–111, 2009.

[12] Kai Renz. *Algorithms and Data Structures for a Music Notation System based on GUIDO Music Notation*. PhD thesis, Technischen Universität Darmstadt, 2002.

[13] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied.* Addison-Wesley, 2001.

[14] Y. Orlarey, D. Fober, and S. Letz. Lambda Calculus and Music Calculi. In ICMA, editor, *Proceedings of the International Computer Music Conference*, pages 243–250, 1994.