In this example, shown in figures 4 and 5, it is possible to listen to either the leader or the followers. The y coordinate of each turtle controls the frequency of one sine wave oscillator, while the x coordinate is mapped to stereo panning. It is also possible to change the minimum and maximum frequencies in the osc/dataMapping box and set different frequency ranges for each breed as well.

Figure 5. Pure data patch for the Pursuit model

In the sonogram of this example, shown in figure 6, the different pursuits and different paths of the leader and followers can be clearly observed. Also, the sonic shape that each oscillator produces can be audibly mapped to the equations governing the behavior of the leader.

Figure 6. Sonogram excerpt for the Pursuit model

5. CONCLUSIONS

We have developed OSC-NETLOGO, a NetLogo extension that allows to create very complex sonic phenomena by taking advantage of NetLogo’s power for designing and building models of complex systems. The extension is very simple to install and use and gives the possibility of mapping any variable of a NetLogo model into an OSC-enabled audio synthesis engine.

We have provided two examples taken from the available models at the NetLogo’s library of models. These examples provide evidence for the capabilities and potential of NetLogo as a sound generating and processing tool. The behavior of complex systems is something that is very appealing from a musical standpoint, and we hope that this tool could be of aid in the efforts of creating new complex sounds and interesting musical material.

6. ACKNOWLEDGEMENTS

This research was funded by Fondecyt Grant #11090193, Conicyt, Government of Chile.

7. REFERENCES


SCOTES LEVEL COMPOSITION BASED ON THE GUIDO MUSIC NOTATION

D. Fober, Y. Orlarey, S. Letz

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

ABSTRACT

Based on the Music Guido 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 (e.g. ‘f’ or ‘g’), 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’, ‘* enum 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><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:
The following GMN code illustrates the concision of the notation. Figure 1 represents the corresponding GUIDO engine output:

```gm
\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 multi-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):

```gm
{ e g f l, a e a l }
```

Figure 2. A multi-voice example

3. GUIDO Calculus

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 `clef, barlines`. For example, the clef operation re-computes 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 piping scores through operations e.g.

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

Here is another example with the `tag` operation applied to the following simple score:

```gm
[clef("f") c d e]
```

Figure 3. Tail operation consistency

Here is another example with the `seq` operation: a raw sequence of

```gm
[clef("g") c d]
```

and

```gm
[clef("g") c e]
```

would give `c d e` and `c d 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:

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

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 GUIDOCalculate. 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 re-computes the notes length but doesn’t affect the time signature or the barlines. For example, the duration operation re-computes 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 piping scores through operations e.g.

```gm
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]
```

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

```gm
[clef("f") c d e] ->[unseen]
```

Table 1. Score level operations

<table>
<thead>
<tr>
<th>operation</th>
<th>args</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>s1 s2</td>
<td>puts the scores s1 and s2 in sequence</td>
</tr>
<tr>
<td>par</td>
<td>s1 s2</td>
<td>puts the scores s1 and s2 in parallel</td>
</tr>
<tr>
<td>rpar</td>
<td>s1 s2</td>
<td>puts the scores s1 and s2 in parallel, right aligned</td>
</tr>
<tr>
<td>top</td>
<td>s1 [ ]</td>
<td>takes the n top voices of s1; when using a score s2 as parameter, n is taken from s2 voices count</td>
</tr>
<tr>
<td>bottom</td>
<td>s1 [ ]</td>
<td>takes the bottom voices of s1 after the voice n; when using a score s2 as parameter, n is taken from s2 voices count</td>
</tr>
<tr>
<td>head</td>
<td>s1 [ ]</td>
<td>when using a score s2 as parameter, n is taken from s2 duration</td>
</tr>
<tr>
<td>end</td>
<td>s1 [ ]</td>
<td>takes the tail of a score after the date d; when using a score s2 as parameter, d is taken from s2 duration 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</td>
</tr>
<tr>
<td>evhead</td>
<td>s1 [ ]</td>
<td>when using a score s2 as parameter, d is taken from s2 duration 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</td>
</tr>
<tr>
<td>tail</td>
<td>s1 [ ]</td>
<td>transposes s1 to an interval r; when using a score s2 as parameter, r is computed as the difference between the first voice, first notes of s1 and s2</td>
</tr>
<tr>
<td>evtail</td>
<td>s1 [ ]</td>
<td>stretches s1 to a duration of r using a ratio r; when using a score s2 as parameter, r is computed from s2 duration</td>
</tr>
<tr>
<td>transpose</td>
<td>s1 [ ]</td>
<td>applies the pitches of s1 to s2 in a loop</td>
</tr>
<tr>
<td>applypitch</td>
<td>s1 s2</td>
<td>applies the rhythm of s1 to s2 in a loop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a)</th>
<th>g \slur(f) (b)</th>
<th>w e c</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.e.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>with</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uncompleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>we'll use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the terms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opened-end</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>to refer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a) form</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opened-begin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for the b) form</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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, \paren`). 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)
  4) putting scores in sequence:
  5) 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.

1) when computing the end of a score, every `pending` repeat must be closed with a repeat end tag.
2) from successive unmatched repeat begin tags, only the first one must be retained.
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 clefs, barlines). For example, the duration operation re-computes 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 piping scores through operators e.g. 

\begin{verbatim}
head x1 | par x2 | transpose "c e 1"
\end{verbatim}

### 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 GUIDOCalculate. 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

<table>
<thead>
<tr>
<th>operation</th>
<th>args</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq x1 x2</td>
<td></td>
<td>puts the scores x1 and x2 in sequence</td>
</tr>
<tr>
<td>par x1 x2</td>
<td></td>
<td>puts the scores x1 and x2 in parallel</td>
</tr>
<tr>
<td>rpar x1 x2</td>
<td></td>
<td>puts the scores x1 and x2 in parallel, right aligned</td>
</tr>
<tr>
<td>open x1 [ ]</td>
<td></td>
<td>takes the n top voices of x1</td>
</tr>
<tr>
<td>bottom x1 [ ]</td>
<td></td>
<td>when using a score x2 as parameter, is taken from x2 voices count</td>
</tr>
<tr>
<td>head x1 [ ]</td>
<td></td>
<td>takes the bottom voices of x1 after the voice n</td>
</tr>
<tr>
<td>evhead x1 [ ]</td>
<td></td>
<td>when using a score x2 as parameter, is taken from x2 voices count</td>
</tr>
<tr>
<td>tail x1 [ ]</td>
<td></td>
<td>takes the head of x1 up to the date d</td>
</tr>
<tr>
<td>evtail x1 [ ]</td>
<td></td>
<td>when using a score x2 as parameter, is taken from x2 duration</td>
</tr>
<tr>
<td>transpose x1 [ ]</td>
<td></td>
<td>id. but on events basis i.e. the cut point is specified in n events count; when using a score x2 as parameter, is taken from x2 events count</td>
</tr>
<tr>
<td>duration x1 [ ]</td>
<td></td>
<td>takes the tail of a score after the date d</td>
</tr>
<tr>
<td>applypitch x1 x2</td>
<td></td>
<td>when using a score x2 as parameter, is taken from x2 duration</td>
</tr>
<tr>
<td>applyrhythm x1 x2</td>
<td></td>
<td>applies the pitches of x1 to x2 in a loop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>applies the rhythm of x1 to x2 in a loop</td>
</tr>
</tbody>
</table>

### 3.2. Notes sequences and segments

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

\{ e a g f l, a e a l \}

### 3.3. 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 \texttt{tail} operation applied to the following simple score:

\begin{verbatim}
[clef"f" c d e]\n\end{verbatim}

A raw cut of the score after 2 notes would give \{a, c\}, removing the clef information and potentially leading to unexpected results (figure 3).

\begin{verbatim}
[clef"f" c d e]\n\end{verbatim} \rightarrow \text{unexpected} \begin{verbatim}
[clef"c" c d e]\n\end{verbatim} \rightarrow \text{expected}

### 3.4. 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 \texttt{dy} for fine positioning of the notation; figure 1 represents the corresponding GUIDO notation; figure 2 shows an example with the \texttt{clef} tags.

\begin{verbatim}
[clef"f" c d e] \par [clef"g" e c] \par [clef"g" e c]
\end{verbatim}

Figure 4. A simple GMN example

### 4. Music notation consistency

In order to solve the notation issues, we propose a simple typology of the notation elements regarding their 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)
  4. putting scores in sequence:
    5. implicit time extent elements starting the second score must be skipped when they correspond to current existing elements.

### 4.1. Notation elements time extent

The GMN format makes a distinction between position tags (e.g. \texttt{clef}, \texttt{notes}) and range tags (e.g. \texttt{ slur}, \texttt{pass}) 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:

- the pending explicit time extent elements must be properly opened (i.e. opened-begin tags, see section 3.2)
- the current implicit time extent elements must be recalled,

- computing the end of a score:
  - the explicit time extent elements must be properly closed (i.e. opened-end tags)
  - putting scores in sequence:
    - 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 \texttt{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 \texttt{pending} repeat end must be closed with a repeat end tag.
- from successive unmatched \texttt{pending} repeat begin tags, only the first one must be retained.
The form:

Table 2. Typology of notation elements.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit</td>
<td>duration is explicit from the notation</td>
<td>slur, cresc.</td>
</tr>
<tr>
<td>implicit</td>
<td>element lasts to the next similar element or to the end of the score</td>
<td>meter, dynamics, key</td>
</tr>
<tr>
<td>others</td>
<td>structure control</td>
<td>coda, da capo, repeats</td>
</tr>
<tr>
<td>-</td>
<td>formatting instructions</td>
<td>new line, new page</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>breath mark, bar</td>
</tr>
</tbody>
</table>

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 open-ended, 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 rest. Note that range tags are viewed as containers and renotations. Let’s first define

score:

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.

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 languages. 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


