INScore expressions to compose symbolic scores

G. Lepetit-Aimon **D. Fober** **Y. Orlarey** **S. Letz**
Grame
Centre nationale de création musicale
Lyon - France
gabriel.lepetit.aimon@grame.fr

ABSTRACT

INScore is an environment for the design of augmented interactive music scores turned to non-conventional use of music notation. The environment allows arbitrary graphic resources to be used and composed for the music representation. It supports symbolic music notation, described using Guido Music Notation or MusicXML formats. The environment has been extended to provided score level composition using a set of operators that consistently take scores as arguments to compute new scores as output. INScore API supports now *score expressions* both at OSC and at scripting levels. The work is based on a previous research that solved the issues of the notation consistency across scores composition. This paper focuses on the language level and explains the different strategies to evaluate score expressions.

1. INTRODUCTION

Contemporary music creation poses numerous challenges to the music notation. Spatialized music, new instruments, gesture based interactions, real-time and interactive scores, are among the new domains that are now commonly explored by artists. Classical music notation doesn't cover the needs of these new musical forms and numerous research and approaches have recently emerged, testifying to the maturity of the music notation domain, in the light of computer tools for music notation and representation. Issues like writing spatialized music [1], addressing new instruments [2] or new interfaces [3] (to cite just a few), are now subject of active research and proposals.

Interactive music and real-time scores are also representative of an expanding domain in the music creation field. The advent of the digital score and the maturation of the computer tools for music notation and representation constitute the basement for the development of this musical form, which is often grounded on non-traditional music representation [4] [5] but may also use the common music notation [6, 7].

In order to address the notations challenges mentionned above, INScore [8, 9] has been designed as an environment opened to non-conventional music representation (although it supports symbolic notation), and turned to real-time and interactive use [10, 11]. It is clearly focused on music representation only and in this way, differs from tools integrated into programming environments like Bach [12] or MaxScore [13].

INScore has been extended with *score expressions* that provide symbolic scores composition features (e.g., putting scores in sequence or in parallel). Building new scores from existing scores at symbolic level is not new. Haskell is providing such features [14]. Freeman and Lee proposed score composition operations in a real-time and interactive notation context [15]. Regarding the score operations used by INScore, they are imported from a previous work [16] that was focusing on the music notation consistency through arbitrary composition.

The novelty of the proposed approach relies on the dynamic aspects of the scores composition operations, as well as on the persistence of the score expressions. A score may be composed as an arbitrary graph of score expressions and equipped with a fine control over the changes propagation.

The paper introduces first the score composition expressions. Next, the different evaluations strategies are explained and illustrated with examples. The articulation with the INScore environment is presented in detail and followed by concrete use cases. A generalization of this approach is finally proposed in the concluding section.

2. LANGUAGE SPECIFICATION

The main idea behind the project is designing a relevant language that provides easy to use tools to compose and to manipulate scores. Indeed, as all the operators have already been defined by the GuidoAR library [16], the point is to imagine a handy way to use them from INScore's scripts.

2.1 The operators

All the operators have a common interface: regardless their actual definition, they always take two scores as input to produce a score as output. The scores are expressed using the Guido Music Notation format [GMN][17]. A few low-level score manipulation operations are defined (which apply perfectly to INScore language's philosophy) with a deterministic behaviour (none of the operators implement random operations). However, the limited number of operators can't be a limitation, as the uniformity between their

operation	arguments	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	$s1$ $s2$	takes as many voices as $s2$ contains from $s1$, starting by the top voice
bottom	$s1$ $s2$	takes as many voices as $s2$ contains from $s1$, starting by the bottom voice
head	$s1$ $s2$	takes the head of $s1$ up to $s2$ duration
evhead	$s1$ $s2$	id. but on events basis i.e. the cut point is specified by $s2$ events count
tail	$s1$ $s2$	takes the tail of a $s1$ after the duration of $s2$
evtail	$s1$ $s2$	id. but on events basis i.e. the cut point is specified by $s2$ events count
transpose	$s1$ $s2$	transposes $s1$ so its first note of its first voice match $s2$ one
duration	$s1$ $s2$	stretches $s1$ to the duration of $s2$
		if not used carefully, this operator can output impossible to display rhythm
pitch	$s1$ $s2$	applies the pitches of $s1$ to $s2$ in a loop
rhythm	$s1$ $s2$	applies the rhythm of $s1$ to $s2$ in a loop

Table 1. INScore operators

inputs and output make them easy to combine into pipelines designs, creating more high-level operators.

See Table 1 for a definition of all the operators.

2.2 Designing a creative language

In the context of software used for artistic creation like INScore, designing a language is not trivial. Like any other creative tools, the *score expressions* language shall inevitably frame the creation process through which the artist must go. To that extent, conceiving a language is actually designing a creative ”work-flow” that the users shall then adopt.

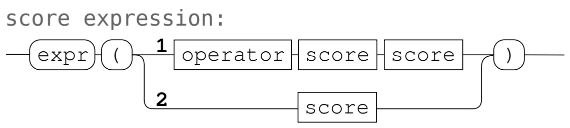
The continuity between inputs and outputs through guido operators allows to compose a music by successively transform and aggregate scores fragments. This process (applying transformations on various materials and combining them into a whole creation) is similar to electro-acoustic creative processes where, after choosing samples, the composer applies effects, equalizer... and mixes them together until the raw musical materials become unrecognisable (il faudrait une référence).

Adapting this approach to the traditional music notation would not only make the language easy to learn for composer (used to the audio pipelines design) but could offer great tools for composition: carving and assembling score samples using structural operators, placing the musical structure in the center of the creative process. In some ways, the art wouldn't emerge from the quality of the raw score fragments but from the process that transforms, shapes, and links them together. Unlike traditional music composition, the structure couldn't only be a classic expected frame any more, but could be the place where the aesthetic lies.

It's with this perspective of audio pipelines and emphasis of the structure that the *score expressions* syntax has been defined. In particular, these expressions should make use of various heterogeneous materials including *score expressions* or existing score objects.

2.3 *Score expressions* syntax

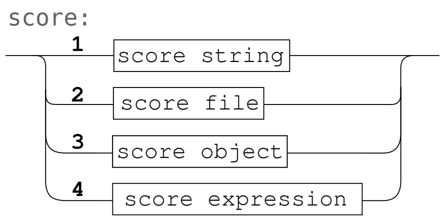
Score expressions can be defined using two syntaxes:



1. The classic syntax reflects the way Guido operators actually work: two scores are combined into one, according to the operator.

2. The alternate syntax defines an expression using a single score, which can be useful to duplicates objects.

Both of the syntaxes make use of `score` arguments. *Score expressions* are quite permissive regarding to their types:



1. `score string`: *Guido* or *MusicXML* string.

2. `score file`: the content of a *Guido* or *MusicXML* file can be imported using absolute or relative path.

3. `score object`: refers to an existing *score object* using a relative or absolute OSC address. *Score object* should be a guido, musicxml or piano-roll object, as well as guido and piano-roll stream object.

4. `score expression`: *Score expressions* can be used as arguments inside *score expressions* (in this case the `expr` token is optional).

Here is an example of a *score expression* that puts a score in parallel with 2 scores in sequence:

```
expr( par score.gmn (seq "[c]" score) )
```

3. EVALUATION SPECIFICATION

The *score expressions* language is first transformed into an internal memory representation. In a second step, this representation is evaluated to produce Guido Music Notation [GMN] [17] strings as output, that are finally passed to the INScore object as specific data.

3.1 Internal representation of *score expressions*

When encountering an *score expressions*, the INScore parser creates a tree representation of it: arguments are stored as leaves and operators as nodes (Figure 1). This tree form allows to easily store, manipulate, assemble and evaluate *score expressions*.

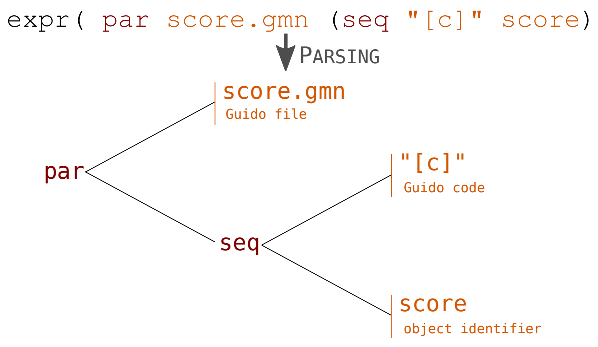


Figure 1. Parsing *score expressions* into tree form.

The tree representation is strictly matching the expression string. Type specification of arguments is the only difference, whereas types are implicit in *score expressions*, arguments are explicitly stored as Guido code or file or identifier... in the tree form.

Once the internal representation has been constructed by the parser, it is stored with the newly-defined object, ready for evaluation.

3.2 *Score expressions* evaluation process

During the evaluation process, the evaluator goes through every nodes of the expression tree using a depth first post-order traversal, reducing all of them into GMN code. A node evaluation is type dependent (Figure 2).
Evaluation of:

- a GMN file gives its content,
- a GMN string returns the string,
- a MusicXML file returns its content converted to GMN code,
- a MusicXML string returns the string converted to GMN code,
- an object identifier gives its GMN code,
- an operator node returns the application of the operator to the GMN code given as parameters.

This evaluation scheme avoid recursion issues (e.g. a score that modifies itself using an expression based on its content) since the caller object is modified only at the end

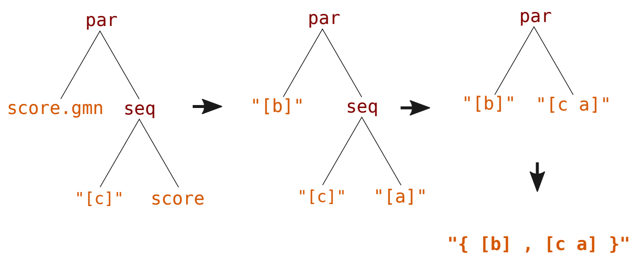


Figure 2. Simple evaluation of an expression tree, where `score` is defined as `[a]` and `score.gmn` contains `[b]`.

of the evaluation process. All arguments are referential transparent by default: each argument is evaluated once and its value is then considered constant.

3.3 Dynamic evaluation of *score expressions*

To ensure that the result is independent, its value is cached after the evaluation so we can retrieve it later without evaluating all the arguments again (which may have changed since). As a result, in the lifetime of an *score expression*, its arguments should only be evaluated once: they are evaluated *statically*. However, pure *static evaluation* is not sufficient. Indeed, if one want to update the value of an expression, he would have to go through the whole evaluation process again. This process can take quite some times which we can't afford in a context of a real-time score viewer.

The *dynamic evaluation* mechanism was introduced to solve this problem. Arguments subject to changes (`score object` or `score file`) can be prefixed with the `&` token, switching them to *dynamic evaluation*. When a re-evaluation is triggered (see 4.2), INScore should check if they have changed, and recompute the expression if so.

To optimize the process, all values (arguments and operators) computed during the evaluation process are cached, this way static branches don't have to be evaluated again and dynamic branches are computed again only if needed (if the cached value of a dynamic argument doesn't match the current one). To be sure that only branches that are likely to change are re-evaluated, the operators nodes are also marked as *dynamically* or *statically* evaluated, according to their arguments. They are considered *static* by default , unless at least one of their arguments is *dynamically evaluated*, in which case they are considered *dynamic*.

4. SCORE EXPRESSIONS API IN INSCORE

In order to fully integrate score operators, the implementation relies as much as possible on INScore existing functionalities. As a result, *score expressions* are fully compatible with INScore's mechanisms, like URL support for arguments, interactive and event support, web support... Of course, some of these features have been enhanced during the implementation process.

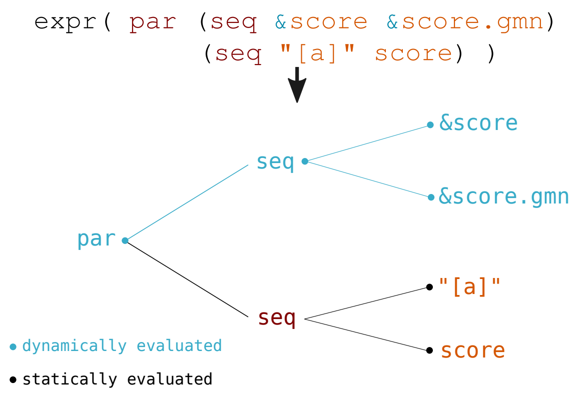


Figure 3. Propagation of dynamic evaluation.

4.1 Declaring *score expressions*

Both `gmn` and `pianoroll` object can be defined with *score expressions* using the standard `set` message. The evaluation of the expression is actually triggered by the new object when the `set` message is processed.

```
/ITL/scene/score set gmn expr(score.gmn);
/ITL/scene/pr set pianoroll expr(&score);
```

The previous example shall create two objects: `score` is a score representation of the music stored in `score.gmn`, while `pr` is a piano roll representation of `score` (here dynamically evaluated due to the `&` prefix).
The declaration of *Score expressions* objects is, like for any other INScore objects, compatible with OSC messages, thus they can also be created remotely through networks.

4.2 *Score expressions* specific messages

Score objects that had been defined using *score expressions* are augmented with new commands.

- `reeval`: Triggers the re-evaluation of the expression tree (check if any of the dynamically evaluated arguments should be updated).
- `renew`: Clears all the stored evaluated values, and re-evaluates the expression. Statically and Dynamically evaluated arguments shall be updated and all the operators are computed again.

All these commands are accessible through the `expr` method:

```
/ITL/scene/score expr reeval;
/ITL/scene/score expr renew;
```

Finally, one can retrieve the *score expression* used to define an object with a `get expr` message:

```
/ITL/scene/score get expr;
```

4.3 Events typology extension

The events mechanism of INScore, which allow to automatically send OSC messages when events are triggered [10] has been extended with a new event: `newData`. It is emitted when the value of any object changes (either because of a `set` or `reeval` message, or because data has been written in a stream object).

Used together with the `expr reeval` message, it makes possible the automation of an expression's re-evaluation when an object changes:

```
/ITL/scene/score set gmn "[a]";
/ITL/scene/copy set gmn expr(&score);
/ITL/scene/score watch newData
    (/ITL/scene/copy expr reeval);
```

In the previous example, changing the value of `score` will automatically change the value of `copy`.

To avoid any recursion problem, `newData` event handling is postponed to the next event loop. As a result, updating the whole scene after changing the value of an object can take several event loop (if one object is referring to another object, itself referring to another one...) and during the process the INScore's graphic scene will certainly go through ephemeral states. However, if objects are defined recursively and are auto-updated using this mechanism, INScore should still be able to update its graphics in time (even if scores are continually changing).

5. COMPOSING SCORE EXPRESSIONS

While the tools already presented allow to compose scores together (including score objects), it is also possible to compose *score expressions* which are stored in these objects using the prefix `~`. Indeed, whereas `score` and `&score` refer to the object's value, `~score` refer to the *score expression* used to define `score`. In practical, before the first evaluation, all arguments prefixed by `~` are replaced by a copy of the expression tree from the corresponding objects.

This tool allows to easily make use of previously defined *score expressions* to create more complex ones.

6. EXAMPLES

6.1 Canon structure

A simple but still well-known music structure is of course the canon. Creating such structure from a `score` is quite easy using the *score expressions* formalism.

```
/ITL/scene/score set gmn
  expr(seq "[a]" &sample);
/ITL/scene/score set gmn
  expr(seq (seq ~score "[b]") ~score);
```

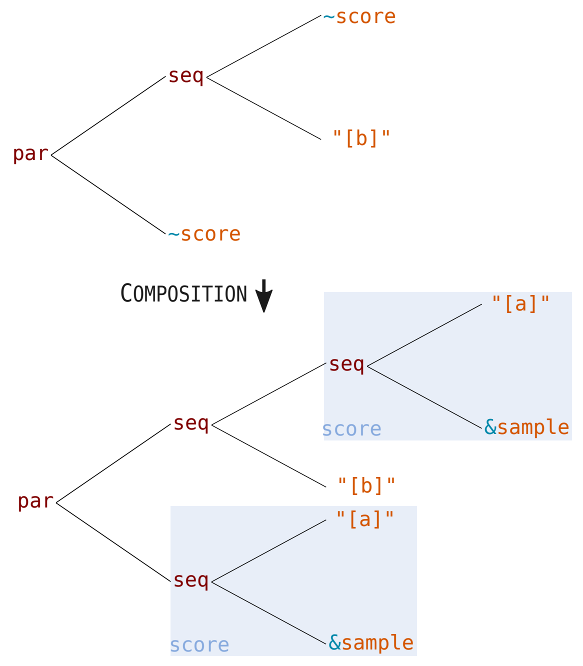


Figure 4. Expanding *score expressions* trees

```
/ITL/scene/score set gmn score.gmn;

# Transposing score
/ITL/scene/canon set gmn
  expr(evtail
    (transpose
      (seq "[c]" &score)
      "[g]")
    "[a]"
  );

# Putting score in sequence with it
/ITL/scene/canon set gmn
  expr(seq &score ~canon);

# Adding a second voice delayed
/ITL/scene/canon set gmn
  expr(par
    ~canon
    (seq "[_/2]" ~canon)
  );
```

The first line import a score into `score`. Then we transpose it to a fifth and adding a second voice delayed of half a measure. Because transposing according to a specific interval is not a basic guido operator, one should combine `transpose` with `seq` and `evtail` to prepend the score with a note, transpose the whole score using this note and finally remove it.

The result is a simple canon:

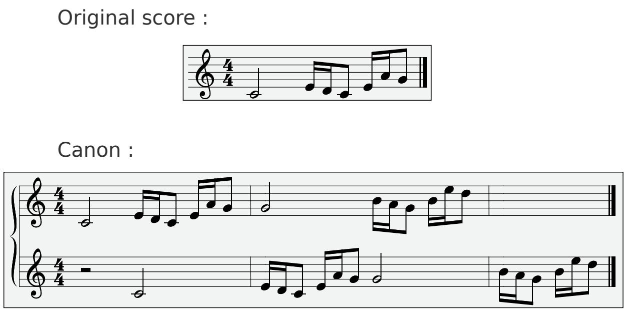


Figure 5. Canon result

6.2 Multiform but still synced scores

Score expressions is a great tools to duplicate and dynamically transform scores, keeping every copies synced to the original.

```
/ITL/scene/origin set gmnf score.gmn;
/ITL/saxo/score set gmn
  expr( evtail
    (transpose
      (seq
        "[e&1]"
        &/ITL/scene/origin )
      "[c2]" )
    "[a]"
  );

/ITL/audience/score set pianoroll
  expr( &/ITL/scene/origin);
```

The previous example create 2 copies of the same score `origin`, one transposed for saxophones and the other show a piano roll version of `origin` used as a visual support for the audience.

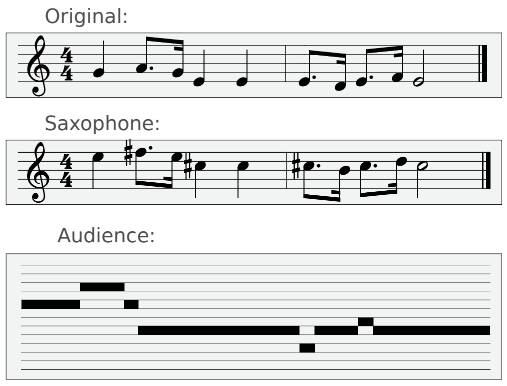


Figure 6. Multiform scores result

7. CONCLUSIONS

Combining the simplicity of GuidoAR operators with the powerful features of INScore (like URL support, full OSC compatibility, interaction support...), *score expressions* fully integrate score composition into the interactive and augmented music score software. They suggest a creative process based upon musical structures and scores aggregation by giving the possibility to compose various score materials including score objects and even themselves. Above all, *score expressions* provides a handy tool to manipulate scores regardless to their origins (files, URL, streams...) or their representations (traditional music notation or piano roll).

Still, the implementation process is not over yet, venturing to the traditional music representation's frontiers. Because traditional notation and piano roll representation is not the only ways to write music, because non-conventional music notation make use of a wider variety of shapes or graphics, the *score expressions* logic should be extended to manipulate any graphical objects of INScore.

Acknowledgments

You may acknowledge people, projects, funding agencies, etc. which can be included after the second-level heading "Acknowledgments" (with no numbering).

8. REFERENCES

[1] E. Ellberger, G. Toro-Perez, J. Schuett, L. Cavaliero, and G. Zoia, "A paradigm for scoring spatialization notation," in *Proceedings of the First International Conference on Technologies for Music Notation and Representation - TENOR2015*, M. Battier, J. Bresson, P. Couprie, C. Davy-Rigaux, D. Fober, Y. Geslin, H. Genevois, F. Picard, and A. Tacaille, Eds. Paris, France: Institut de Recherche en Musicologie, 2015, pp. 98–102.

[2] T. Mays and F. Faber, "A notation system for the karlax controller," in *Proceedings of the International Conference on New Interfaces for Musical Expression*. London, United Kingdom: Goldsmiths, University of London, June 2014, pp. 553–556. [Online]. Available: http://www.nime.org/proceedings/2014/nime2014_509.pdf

[3] W. Enstr"om, J. Dennis, B. Lynch, and K. Schlei, "Musical notation for multi-touch interfaces," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, E. Berdahl and J. Allison, Eds. Baton Rouge, Louisiana, USA: Louisiana State University, May 31 – June 3 2015, pp. 83–86. [Online]. Available: http://www.nime.org/proceedings/2015/nime2015_289.pdf

[4] R. R. Smith, "An atomic approach to animated music notation," in *Proceedings of the First International Conference on Technologies for Music Notation and Representation - TENOR2015*, M. Battier, J. Bresson, P. Couprie, C. Davy-Rigaux, D. Fober, Y. Geslin, H. Genevois, F. Picard, and A. Tacaille, Eds. Paris, France: Institut de Recherche en Musicologie, 2015, pp. 39–47.

[5] C. Hope, L. Vickery, A. Wyatt, and S. James, "The decibel scoreplayer - a digital tool for reading graphic notation," in *Proceedings of the First International Conference on Technologies for Music Notation and Representation - TENOR2015*, M. Battier, J. Bresson, P. Couprie, C. Davy-Rigaux, D. Fober, Y. Geslin, H. Genevois, F. Picard, and A. Tacaille, Eds. Paris, France: Institut de Recherche en Musicologie, 2015, pp. 58–69.

[6] R. Hoadley, "Calder's violin: Real-time notation and performance through musically expressive algorithms," in *Proceedings of International Computer Music Conference*, ICMA, Ed., 2012, pp. 188–193.

[7] ——, "December variation (on a theme by earle brown)," in *Proceedings of the ICMC/SMC 2014*, 2014, pp. 115–120.

[8] D. Fober, Y. Orlarey, and S. Letz, "Inscore – an environment for the design of live music scores," in *Proceedings of the Linux Audio Conference – LAC 2012*, 2012, pp. 47–54.

[9] ——, "Augmented interactive scores for music creation," in *Proceedings of Korean Electro-Acoustic Music Society's 2014 Annual Conference [KEAMSAC2014]*, 2014, pp. 85–91.

[10] D. Fober, S. Letz, Y. Orlarey, and F. Bevilacqua, "Programming interactive music scores with inscore," in *Proceedings of the Sound and Music Computing conference – SMC'13*, 2013, pp. 185–190. [Online]. Available: fober-smc2013-final.pdf

[11] D. Fober, Y. Orlarey, and S. Letz, "Representation of musical computer processes," in *Proceedings of the ICMC/SMC 2014*, 2014, pp. 1604–1609. [Online]. Available: inscore-processes-final.pdf

[12] A. Agostini and D. Ghisi, "Bach: An environment for computer-aided composition in max," in *Proceedings of International Computer Music Conference*, ICMA, Ed., 2012, pp. 373–378.

[13] N. Didkovsky and G. Hajdu, "Maxscore: Music notation in max/msp," in *Proceedings of International Computer Music Conference*, ICMA, Ed., 2008.

[14] D. Quick and P. Hudak, "Grammar-based automated music composition in haskell," in *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling and design*, ser. FARM '13. New York, NY, USA: ACM, 2013, pp. 59–70. [Online]. Available: http://doi.acm.org/10.1145/2505341.2505345

[15] S. W. Lee and J. Freeman, "Real-time music notation in mixed laptop-acoustic ensembles," *Computer Music Journal*, vol. 37, no. 4, pp. 24–36, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1162/COMJ_a_00202

[16] D. Fober, Y. Orlarey, and S. Letz, "Scores level composition based on the guido music notation," in *Proceedings of the International Computer Music Conference*, ICMA, Ed., 2012, pp. 383–386. [Online]. Available: icmc12-fober.pdf

[17] 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.