OMLILY: FILLING THE NOTATIONAL GAP BETWEEN COMPOSITION AND PERFORMANCE

Karim Haddad, Carlos Agon

OMLILY: FILLING THE NOTATIONAL GAP BETWEEN COMPOSITION AND PERFORMANCE

Karim Haddad
IRCAM
karim.haddad@ircam.fr

Carlos Agon
IRCAM - UPMC
carlos.agon@ircam.fr

ABSTRACT

This paper describes the design, development, usage, limitations and prospects for future development of Omlily, an OpenMusic library for editing scores with Lilypond, using OpenMusic musical editors [1] .

1. INTRODUCTION

Using a Computer Assisted Composition (CAC) environment such as OpenMusic (OM) [1], rich in functions, macros, and algorithms for composition, we assemble a huge amount of musical material, such as pitch, rhythm, and other musical structures contained in OpenMusic musical classes and editors. OpenMusic editors are powerful objects, they can deal with the most complex musical structures. However, they also have certain limitations regarding display and typesetting capabilities. Furthermore, editing scores directly in these editors seems to be very laborious when particulary if the pieces are of long duration. This is due to two important factors that could be considered as flaws (or weaknesses) in OpenMusic: lack of efficient editing tools and slowness in the display time. Another essential element not available to the composer in the OpenMusic environment, but also related to display, is the music sheet layout view. Although this option is present, the score displayed has very few (if any) options for layout. In order to have a presentable draft for the composer to work with, we can benefit greatly from Lilypond's extraordinary typesetting layout features [2] .

Writing a fully featured typesetter and viewer from scratch in OpenMusic involves coding it in CommonLisp [2]. This does not seem to be a bright approach for this issue at all. In the past we have developed internal Lisp code in OpenMusic to export OpenMusic musical objects to commercial typesetting programs based on their own private SDK standard format. We have also written code for MusicXML standard format export. However the validation of this standard format does not seem stable as a standard due to private parsers of each commercial typesetting software. Therefore, we decided to use a pre-existing typesetter, Lilypond is a GNU opensource typesetting software [3], that is maintained up to date continually. We have based our exchange code on Lilypond's own syntax. Moreover, Lilypond as MusicXML, are widely used in other CAC environment such as PWGL, FOMUS, Orchids, Abjad, etc... Lilypond seems be the best choice and solution regarding rendering, efficiency and notational possibilities. OpenMusic's architecture, conceived as a modular environment, handling and loading only essential features, and following the user's needs and requirements, will welcome such a library, typesetting being the most requested and necessary feature for OpenMusic's end user.

[1] https://github.com/karimhaddad/omlily/

[2] such as paper, tuplet, even color display

2. USAGE

2.1 General usage

The purpose of this library is to combine both of the potentialities of a CAC environment by devising complex forms of compositions with the editing efficiency of a powerful typesetter in a dynamic form of interaction. Moreover, another aim will be to require minimal effort for producing huge and complex input content such as rhythm, pitch and other musical material in the form of a typesetting document. In another sense, the composer, most of the time, will need to go back and forth from a CAC environment to typesetting and vice versa most of the time managing huge amounts of musical data, particularly in the case of big ensemble or orchestral compositions.

If we schematize the work-flow of the different steps of a typical compostional process using CAC tools, we can describe it as follows:

- The pre-compositional stage, which involves automatic or algorithmic computation, sound analysis, combinatory computations, etc., in other words, the first draft of musical material production;
- The first stage of typesetting: pre-editing, previsualization, in order to rearrange and revisit the material as a score sheet. This might lead to adjustments, pre-instrumentation, voice redistribution, form editing, etc.;
- The intermediate feedback phase in CAC for re-computation, corrections, arrangements, modifications, form re-injections, segmentation,etc.; this step is the most dynamic one;

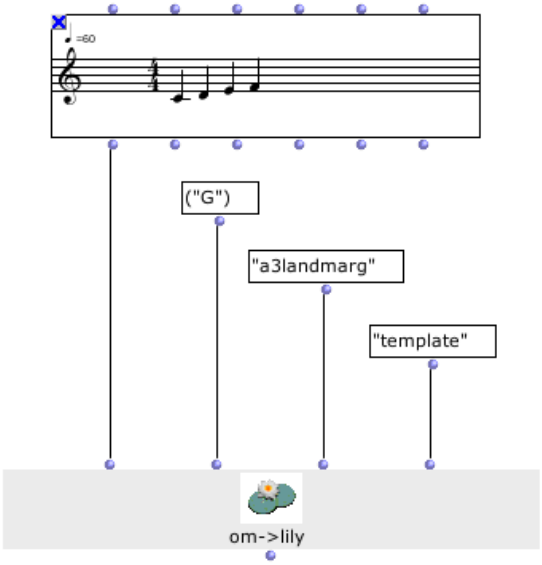

Figure 1. Exporting from OpenMusic to Lilypond

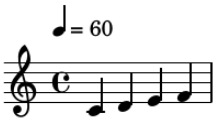

Figure 2. Lilypond rendering

- Finalization of typesetting by Lilypond (final score engraving).

2.1.1 *Exporting Lilypond files from OpenMusic*

This is done using the *om → lily* method (see Fig.1). It handles VOICE, POLY, and CHORD-SEQ OpenMusic objects. Four arguments are given to this method :

- *self*: the OpenMusic object to translate into Lilypond;
- *clef*: the clef needed (it could be a list of clefs in the case of a POLY object);
- *paper*: the paper default or user's template;
- *layout*: the default or user's score context (equivalent to notational preferences).

Once the .ly file is written, OpenMusic will redirect the file to Lilypond present binary and compile the file (Fig.2) opening it with the user's preferred PDF reader [3] .

2.1.2 *Importing Lilypond files to OpenMusic*

Each exported score from OpenMusic will generate a file where a commented code will be included. This code (cf. Fig 3), once uncommented, and recompiled with Lilypond binary, will create a Scheme translation file of the score. By default it will be named *temp.lisp*.

```
% #(with-output-to-file "temp.lisp"
% (lambda () #{ \displayMusic {
<<
\new StaffGroup
 << non unecessary
\new Staff  {
\one
}
 >>
 >>
% } #}))
}
```

Figure 3. Lilypond's compilation instructions for a Scheme transcript of a .ly file

Lilypond will then generate a Scheme code where all of the musical and layout elements are translated by the *make-music* method. Again, once this file is evaluated with the *lily→om* method in OpenMusic, the necessary data will be translated and instantiated into an OpenMusic editor (Fig. 4)

2.2 Particular usage

2.2.1 *Polymetrics and polytempi notation*

OpenMusic has the ability to display and perform the most complicated and sophisticated rhythmical expression due to the implementation of Rhythm Trees (RT) [4]. This includes embedded tuplets, polymetric music, polytempi, and irrational measures [4] such as 3/21 or 4/10 for example (Fig. 5). Such cases are handled poorly or at all, by most commercial typesetters since these features are seen as completely experimental and related specifically to contemporary compositional practice, and therefore not popular for most users and musicians; as a result, they are not supported in these environments. Fortunately, Lilypond has the ability to deal with most, if not all, of these issues, with a standard approach. However in cases such as polymetric music, one should rely upon a slightly different syntax in order to display the complex polyphony correctly.

When using polymetrics in standard notation, i.e with binary time signatures, Lilypond will automatically display the correct score. Hence, the *om→lily-gen* method will be used. If the score uses non-standard, non-binary time signatures, the user should choose the *om→lily-spec* method. Both methods are included in the generic method *om→lily-gen*, which will automatically depict the presence of polymetric time signatures.

[3] The Lilypond binary and version number are generated automatically once the library is loaded in OpenMusic. However, the user may search for the desired Lilypond and PDF reader binaries in OpenMusic's "External" preferences tab.

[4] cf. Livre Premier de Motets: The Time-Block Concept in OpenMusic[5]

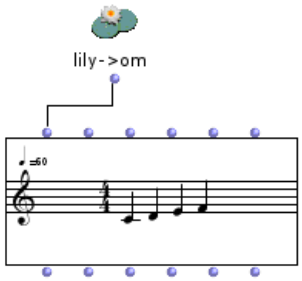

Figure 4. Importing form Lilypond to OpenMusic

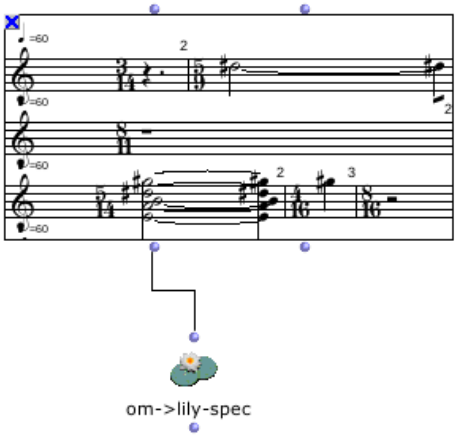

Figure 5. Polymetric score exportation

Although polyphonic polymetric notation is possible in Lilypond, it is not fully documented for the case of "irrational" time signatures. The scaling factor of duration is calculated as follows :

For instance, if we consider a measure with a time signature of 3/20 (this will be a measure of three sixteenth note of a quintuplet), we will multiply all sixteenth notes figures by 1/5 (note 16*1/5). For binary time signatures, the multiplication factor would invariably be 1/4.

The calculation is made using the `calc-scale-fact` function (Fig.7). This will first calculate the beat-symbol using the `find-beat-symbol` function (the note figure, e.g. quarter note, eighth note, etc...) that the denominator of the time signature is related to. For instance, if we have a time signature of 4/12, twelve will refer to a twelfth of a whole note, equaling an eighth note of a triplet [5]. The factor will be then calculated with this formula: $\frac{(beat-symb/4)}{denom}$ = $\frac{8/4}{12}$. The scaling factor in our example will therefore be equal to 1/6.

2.2.2 *Discrete spanner notation*

One immediately see that this library is addressed mainly for written instrumental compositions. However, in prospect, there is also to be some audio/graphical symbolic notational outputs that can be extracted from break-point functions (BPF), or representation of audio sources. Indeed, some OpenMusic classes have the ability to enclose both representations, symbolic and graphical outputs of musical objects. Fig.8 shows an example of a Lilypond score rendering exported from OpenMusic's BPF objects. `make-music` Scheme function

[5] 1/3 of a whole note = a half note figure. 1/6 = a quarter note, etc...

3. IMPLEMENTATION

OpenMusic Lisp code and Lilypond Scheme [6] syntax handle most of the communication between the two environments. We can describe two different levels for both environments: the first level is proprietary and typical for each (OpenMusic patches, as a visual programming language, or VPL); and a latex-like scripting language, a syntax for editing Lilypond .ly files. Consequently, each of these levels remains opaque to each others. It is on an internal level that the communication occurs: the CommonLisp language used by OpenMusic's kernel, and the Scheme expressions which are part of Lilypond's Guile interpreter. This is where the link between both environments lies, due to the fact that both Scheme and Commonlisp are dialects of the Lisp language.

We can summarize this communication schematically with a straightforward process :

OM patch (VPL)→(CommonLisp) RT→Lilypond (ly syntax) →Scheme interpreter→OM musical class object editor

An example of this translation : A simple voice bearing a single note (Fig.9), such as a middle C on a G staff [6] in a quarter note figure in a 1/4 time signature will be written [7] in OpenMusic as :

```
(make-instance 'voice
:tree '(1/4 (((1 4) (1))))
:chords '((6000))
:tempo '(60)
)
```

Skipping the paper settings, page layout and contexts, the translation of this simple object will render this Lilypond code :

```
...

"one"=
{
\tempo 4 = 60
\time 1/4
c'4
|
}

\score {
 {
% #(with-output-to-file "temp.lisp"
% (lambda () #{ \displayMusic {
<<
```

[6] In OpenMusic, the staff keys are not explicitly formulated.

[7] We may notice that in OpenMusic, the rhythmical inforamtion is separated from pitch information.

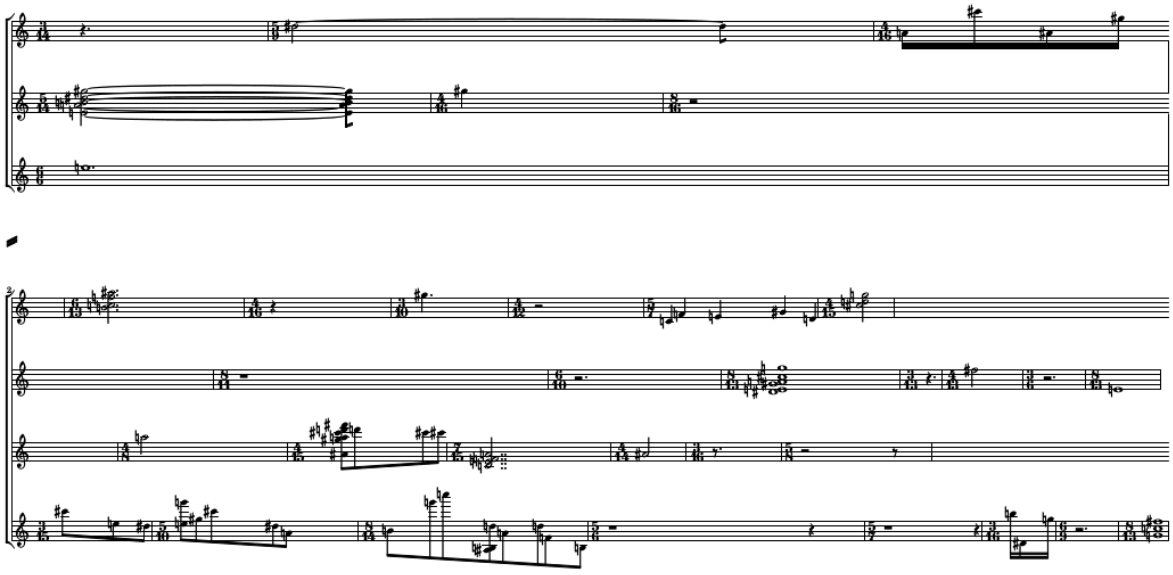

Figure 6. Polymetric score rendering

```
(defun calc-scale-fact (time-signature)
(let*
((denom (second time-signature))
     (beat-symb (find-beat-symbol denom))
     (fact (/ (/ beat-symb 4) denom)))
 (if (= 1 fact) 1/4 fact)))
```

Figure 7. Calculating scaling factor of each note figure according to its time signature.

```
\new StaffGroup
 <<

\new Staff  {
\one
}
 >>
 >>
% } #}))

}

...
```

We can already observe what new data has been produced in this translation:

- Paper settings, layout and contexts (omitted here).
- Alteration display rules.
- Staff grouping layout (not shown here, since it is a single voice).

The generated data will grow more and more as we progress toward typesetting. This is due to the required paper settings, staff layout, line breaks, etc. Inversely, from Lilypond to OpenMusic, most of the typesetting content will be omitted and filtered to the strict minimum since these will not be necessary for the instantiation of an OM object, which editor is a set of linear display of graphical notation using fonts and line drawings without page layout.

The intermediary Scheme translation code generated from OM using the *lily→om* method will look like this:

```
(make-music
...


(make-music
 'TimeSignatureMusic
   'beat-structure
   '()
   'denominator
    4
   'numerator
    1)

 (make-music
  'NoteEvent
      'duration
     (ly:make-duration 2)
      'pitch
      (ly:make-pitch 0 0))

 (make-music
   'BarCheck)
...
)
```

After retrieving pagination, layout, and some unneeded information [8] , if we carefully examine the reduced Scheme

[8] We have omitted some headings in order to save some place in our paper. However, we have displayed intentionally most of the required essential notational data.

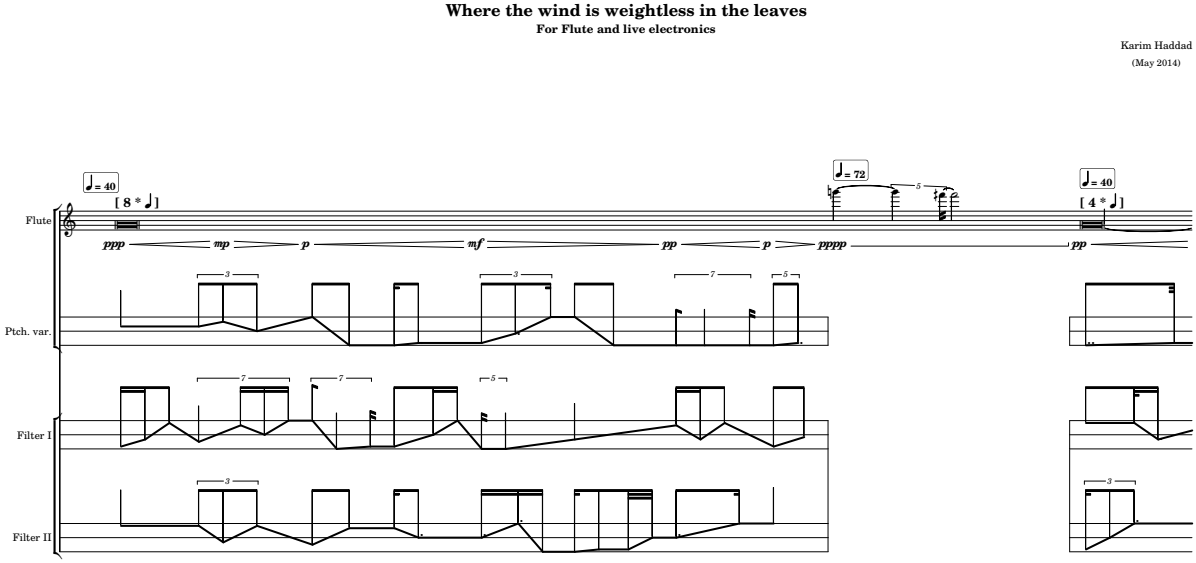

Figure 8. Control spanners

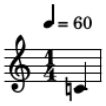

Figure 9. Single note

code above, carefully we will distinguish three redundant and distinct calls of the `make-music` Scheme function. This function is "for internal use, which is the preferred interface for creating music objects"[7]. These objects are created in Lilypond by C++ code, and represent a hierarchy of instances of musical notation. Thus in our example, our instances will be the three arguments of `make-music` function 'TimeSignatureMusic, 'NoteEvent, and 'BarCheck.

In order to translate these instances into OpenMusic compliant objects, we have transformed the `make-music` Scheme function into a CommonLisp (CLOS) method. This method will in turn, instantiate the different classes, such as staff, pitch, duration, etc. needed for OpenMusic to construct a compliant object according to each given type. Inspecting the previous example, we will again find our initial data unaltered.

Here is an example of the `make-music` Scheme function translation in CommonLisp regarding pitch and rhythm transcription [9] :

```
(defmethod make-music
((type (eql 'NoteEvent))
         &rest other-args)
(if *lil-imp-pitch*
  (let* ((art (car
           (find-value-in-lily-args
           other-args
       'articulations)))
             (tie
              (if (equal 'tieevent art)
              1 0))
             (durs
              (find-value-in-lily-args
              other-args 'duration))
             (fig (car durs))
             (dot (second durs))
             (fact (third durs)))
        (make-instance 'lily-dur
              :figure fig :dot dot
              :fact fact  :tieevent tie
                          :restevent 0 ))
    (if (not (member 'tieevent
     (flat (find-value-in-lily-args
     other-args 'articulations))))
        (remove nil (list
         (find-value-in-lily-args
         other-args 'pitch)))  )))
```

However, we should point out that only data necessary for OpenMusic's objects instantiation will be taken into account. If the Lilypond file has been modified by including extra notations such as dynamics or text markings, the import procedure will ignore these. Round tripping is something which will be included in the future as a standard

[9] In Lilypond, time signature is a separate piece of information (as we may have seen in the intermediate Scheme translation given above), which is not the case with OpenMusic's RT structure[4], where it is completely integrated into rhythm information. We are not displaying the method concerning it per se.

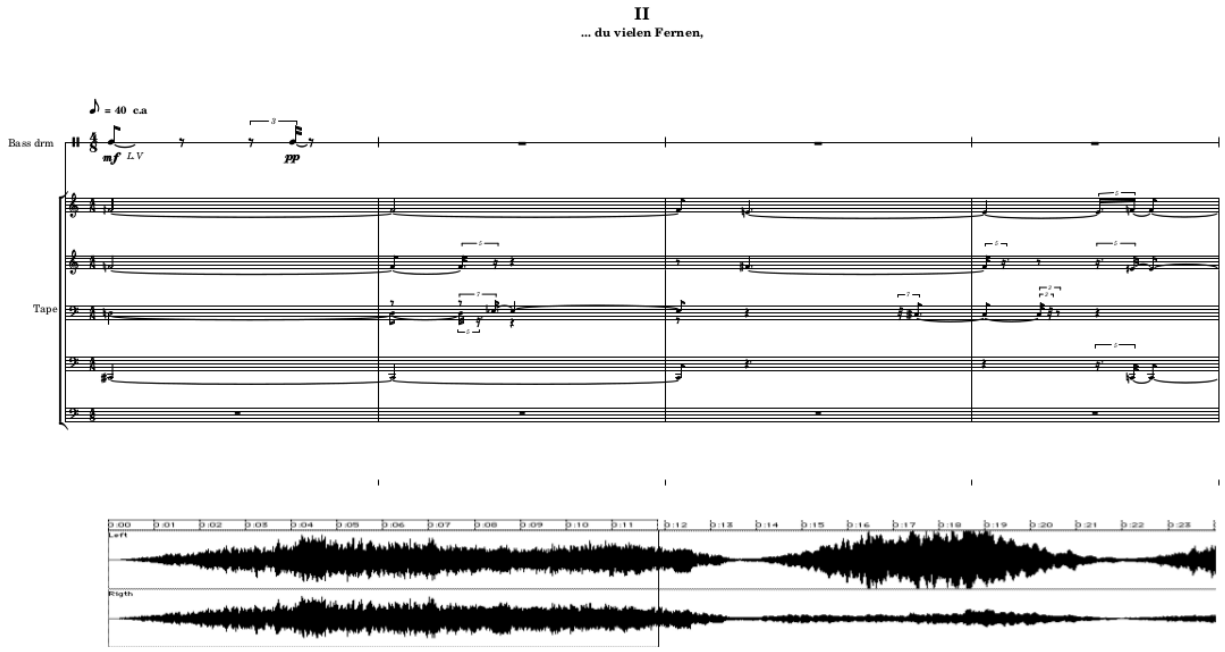

Figure 10. Score with audio wave shape

procedure (cf. future development section).

4. LIMITATIONS

Modern score setting is a field that embraces rich figuration and symbolic notational representation. It would likely be an extremely difficult task to encompass the majority of the musical symbolic representations necessary to render them in such a varied context. As indicated above, most of the essential hierarchical musical classes are represented with the inclusion of independent features such as polytempo; embedded and recursive rhythm structures; dynamics; linear spanners with the exclusion of musical elements not yet supported by OpenMusic, such as grace notes [10]; lyrics; crescendos/diminuendos; and other continuous symbolic extra notation features.

In Lilypond, page breaking process is an implemented algorithm. This performs well with strictly measured music. However, exporting page turns from OpenMusic is not supported, since in this environment, no such concept exists. It is based on graphical display edits rather than on rational musical ones. The main issue will therefore be the page setup regarding the segmentation of a printed output. In the case of graphical representation such as audio amplitude profile embedded in the score, as in Fig.10 for instance, the page layout will be determined by the graphical representation itself. This task normally left to the typesetter's discretion (weight of notes by page, performer breaks, etc ...) cannot be automated in any way, since it is based on a performer-composer appreciation mostly by typesetting rules with regard to a graphical layout that will constrain the page settings.

For instance, if we examine Fig.6 closely, we will see that the first system has a greater line span than the second one which is more dense than the former. Here, the line breaks were calculated by Lilypond without any explicit instructions on the part of the user. For the time being with the absence of a graphical interface within OpenMusic, for such a task we would have to, if needed, edit the breaks in Lilypond manually.

5. FUTURE DEVELOPMENT

Certain immediate issues are under imminent implementation. We can list some of these here:

- CHORD-SEQ structure import from Lilypond to OpenMusic; [11]
- PianoStaff support, most particularly cross stemming and automatic voice splitting;
- Graphical interface for page layout and other typesetting preferences.

Apart form these peripheral additions, our goal for an innovative implementation of this inter-exchange program would ideally be to achieve complete interaction between the two environments, such as musical computation interacting directly with the resulting typesetting. This could

[10] Grace notes are not yet supported graphically. However they are integrated as objects. There is an ongoing effort to implement them in our recent development of a notational viewer which will be hopefully integrated into OpenMusic.

[11] The Lilypond-to-OpenMusic import feature works only for a simple general usage for the time being. It does not yet support the CHORD-SEQ export, since it is not metered music and requires a different approach.

be achieved by automatic computation, i.e. building routines which will deal with the given data (musical material/typesetting material) automatically in order to have direct rendering and transformation on both sides of the platform, e.g. changing a section in the editor (here, Lilypond) and feeding this back in an OpenMusic patch or vice-versa. Ideally, the goal is to build an embedded graphical editor in OpenMusic as stated before, containing Lilypond typesetting rendering with a Lilypond-syntax-embedded editor.

With this in perspective, we are planning to implement an intermediate inter-exchange format for round tripping. This will allow for the safe keeping of all incoming and outgoing data to and from OpenMusic and Lilypond. In order to achieve this objective, we should survey most of the existing classes and internal properties of Lilypond making them available in a registry that will be compliant with both platforms.

Such a thing is possible, even recommended, since the SHEET object[8] (see Fig.11), still in beta state, should be finalized in such a way.

SHEET[8] is a graphical OpenMusic editor whose purpose is to have abilities in editing and "throwing" a computational operation such as transposition or any other serial operation on symbolic objects such as measures, groups, chords, rhythm, etc. It has the capacity to display symbolic notation (scores), audio wave forms, and Break Point functions (BPF) along with OpenMusic patches.

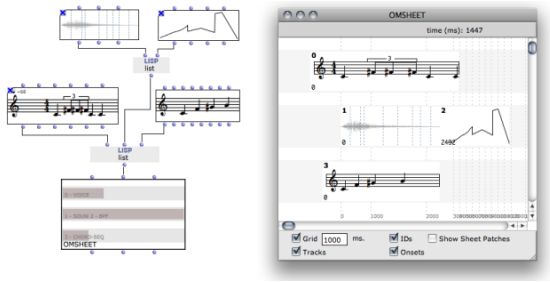

Figure 11. OpenMusic SHEET editor object

6. CONCLUSIONS

Omlily OpenMusic's library seems for us to be, for the time being, a very good solution for musical score interchange between OpenMusic and Lilypond. It is also a powerful research tool for experimentation for a new scope of musical ideas due to its unique potentialities in exploring complex musical structures. We have been using it for sometime now and have conceived many compositional scores with it, from solo to ensemble music. Extending its potentialities in order to enclose more notational data seems promising. This interchange can lead to expanding both environments, having, on one side, the ability to integrate CAC functions and computations in the Lilypond environment, and on the other, one of the best typesetters for rendering scores in OpenMusic.

Acknowledgments

We would like to thank Jean Bresson for his cooperation, Jeremy Coffman and Deborah Lopatin for their proofreading and last but not least, Gerard Assayag for his valuable expertise and advice.

7. REFERENCES

[1] C. Agon, "Openmusic : Un langage visuel pour la composition musicale assiste par ordinateur," Ph.D. dissertation, IRCAM - Univ. Paris 6, dcembre 1998.

[2] G. Steele, *Common Lisp the Language, 2nd edition*. Digital Press, 1990.

[3] *LilyPond, a system for automated music engraving*. Firenze, Italy: Colloquium on Musical Informatics, 2003.

[4] C. Agon, K. Haddad, and G. Assayag, "Representation and rendering of rhythmic structures," *WedelMusic Darmstadt*, vol. 25, pp. 109–113, 2002.

[5] K. Haddad, *Livre Premier de Motets: The Time-Block Concept in OpenMusic*, 1st ed., ser. The OM composer's book. Edition Delatour, 2008, vol. II.

[6] G. J. Sussman and G. L. S. Jr., "Scheme: An interpreter for extended lambda calculus," in *MEMO 349, MIT AI LAB*, 1975.

[7] *Lilypond - Internals Reference*, 2000-2012.

[8] J. Bresson and C. Agon, "Scores, programs and time representations:the sheet object in openmusic," *Computer Music Journal*, vol. 32, no. 4, 2008.