ENP note head designer

Mika Kuuskankare, Mikael Laurson

ENP NOTE HEAD DESIGNER

Mika Kuuskankare
Department of Doctoral Studies
in Musical Performance and Research
Sibelius Academy
Finland
mkuuskan@siba.fi

Mikael Laurson
Centre for Music and Technology
Sibelius Academy
Finland
laurson@siba.fi

ABSTRACT

This paper presents a new visual editor called ENP Note Head Designer (henceforward ND). ND is aimed at assisting both novice and expert users to design custom note heads in ENP. Both visual and text-based interfaces are provided. Using the latter approach it is possible to take advantage of the power of Common Lisp, CLOS, and OpenGL (the base languages of ENP). The former approach, in turn, allows to use a set of graphical tools to define the note heads visual appearance. It is also possible to mix these approaches. Furthermore, the note heads are viewed and designed in an actual musical context. These concepts make the current system unique compared to the existing approaches.

1. BACKGROUND

ENP Note Head Designer (ND) is part of an increasing collection of visual tools inside ENP [3]. Currently, these tools include (1) Color Editor, (2) Expression Designer, and (3) Note Head Designer. The purpose of these tools is to provide a visual interface for constructing complex music notation related CLOS objects and to provide assistance to those users that do not possess the needed programming skills to use the strictly text-based approach. It is important to realize, however, that the visual approach is not exclusively for the unexperienced users. Some objects may be realized more easily by using a purely graphical approach while the others may be more suitable to be defined using a programming language. Thus, a text-based approach is provided along with the graphical one.

Uniform design principles have been applied to all aforementioned tools. The tools share same kinds of editors, components and functionality which makes them easier to learn. Most editors also contain an interactive feedback view. This makes it possible to design objects in their native environment, e.g, the designer sees the note head as it would appear in the score. Furthermore, visual synchronized feedback is one of the most important design concepts behind ENP. Every applicable GUI operation behaves this way and thus all ENP tools share the same design principles.

The use of visual editors in our case has several advantages. First, the underlying CLOS syntax can be hidden to allow the user to concentrate on the task in hand (e.g., designing note heads). Second, following the purely text-based scheme involves routinely repeating several mandatory low level program components (i.e., class definitions, specialized method definitions, etc.). This kind of repetitive work load is considerably reduced. Finally, this approach is also less prone to errors since the syntax and most of the underlying CLOS code is handled automatically.

Most of the existing commercial notation programs, such as Finale[1] or Sibelius[8], provide some kind of interface for the user to change the appearance of the note heads. The most fundamental and the most frequent approach is that the user can select a note head character and a font typeface. On the other hand, Finale additionally provides some graphical tools allowing to define the note head appearance by drawing arbitrary shapes in a graphics canvas. However, in this case the note heads are designed without the presence of any musical context. In Common Music Notation (CMN, [11]) it is also possible to define note heads algorithmically by using Common Lisp. Finally, LilyPond [9] allows to change the note head types algorithmically, by using Scheme programming language.

ND is a visual tool that allows to design and view user-definable note heads in ENP. ND provides an approach that mixes both the power of textual programming and the use of traditional graphical tools. All the needed program components and user-interface components (e.g., menus) are prepared automatically by ND. The note heads are also viewed and designed in a musical context. Moreover, the designed note heads are ready to be used in the current ENP session and can be saved in a file to be loaded either automatically or by demand.

In the following sections we first describe the note head scheme used in ENP. Then we present the ND window and its components. After this we examine in detail the programming interface. Finally, we give a comprehensive set of examples prepared with the help of ND. The paper ends with some concluding remarks.

2. THE ENP NOTE HEAD SCHEME

In this Section we describe the internal representation of note heads in ENP. We enumerate the different types of note heads and also give examples of their visual counterparts. Internally the note heads can be represented in the

following ways:

(1) Strings or characters. In this case the object is usually the literal representation of the note head, i.e., the string or character is drawn as is using the default music notation font. The set of standard ENP note heads are defined in this way. Figure 1 gives a set of some character-based note heads in ENP.

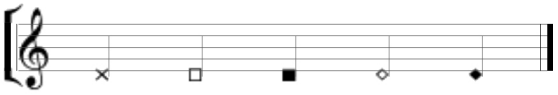

Figure 1. Character-based note heads in ENP.

(2) Keywords act as symbolic references to specific CLOS methods that, in turn, draw the note heads according to the provided code. This approach is suitable for more complex and usually, but not necessarily, static note heads, which cannot usually be represented with a simple character or string. Figure 2 gives an example of one such note head, denoting a tambura effect used in guitar music.

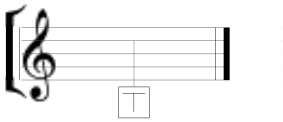

Figure 2. A tambura note head.

(3) Objects. When note heads are expressed as objects every note head can have a different visual appearance and an individual set of data and properties. This allows, for example, to use different kinds of user editable shapes as note heads.

The internal *bpf note head* is an example of this kind of an object (see Figure 3). In this case the note head is represented as a break-point function. It can be used for several different purposes, among others, to define the visual appearance of the note head itself by providing the note head shape or allowing the user to define one, or to contain control information for playback (e.g., envelope). The break-point function is also fully editable through the GUI.

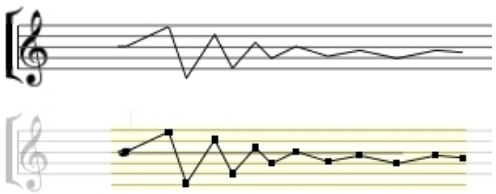

Figure 3. Two bpf note heads. The latter one is in an editable state with the movable break-points shown as black rectangles.

The note heads constructed with ND usually belong to the category (1) or (2). The third option (note head objects) relies on inheritance and this is not yet supported by ND.

3. THE ENP NOTE HEAD DESIGNER

There are only a handful of things that need to be defined in order to create a new note head using ND. The required minimum is: Unique name, which in this case is a Lisp keyword (e.g., :tambura, :sprechstimme, etc.), and the note head definition code in Lisp and/or hand drawn graphics composed out of the provided graphical primitives. All additional code and the UI components, i.e, classes, methods, and contextual menu entries, are provided automatically by ND.

3.1. The Components of the Note Head Designer

ND is represented by an ND window, which, in turn, contains several specialized components. As can be observed in Figure 4, on the top of the window there is a properties view (1) containing inputs (e.g., note head name and font). Next there is a code view (2) that can be used to enter the note head definition in Lisp. At the bottom of the window there is a graphical tool palette (3), and a preview score/graphics canvas (4). The preview score is fully editable. Additionally, there are also some push buttons to deal with the preview score and the note head definition source code.

The ruler above the staff entitled 'width' shows the extent of the note. This can be used as a visual aid when drawing the note head definition by hand.

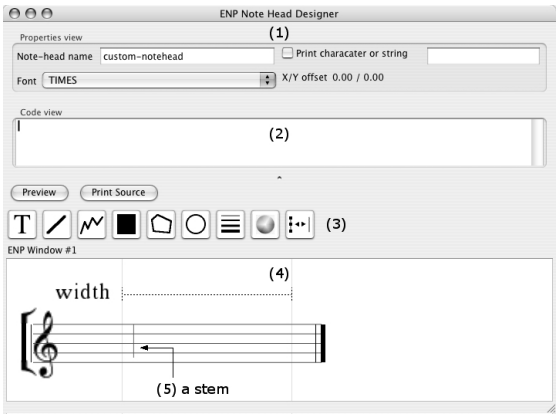

Figure 4. The components of the ND window. (1) the properties view, (2) the code view, (3) the graphical tools, and (4) the preview score/graphics canvas. The ruler (labeled 'width') showing the extent of the note is displayed above the staff. Also, (5) shows the stem of the chord the designed note is associated with.

There are basically two ways to define a note head in ED. First, by using the text-based or code-based approach the user can define the the note heads appearance with the help of the provided set of drawing commands and special variables. Second, drawing by hand using a special graphics canvas and a set of graphical tools.

3.2. Code View

In the code view the user can write standard Lisp code (see Figure 4). The users can utilize any of the OpenGL [12] functions provided by LispWorks. Additionally, ENP provides also a set of graphical primitives, e.g., draw-2D-arrow, draw-2D-box, draw-2D-bracket, draw-2D-circle, draw-2D-line, draw-2D-lines, draw-2D-point, draw-2D-polygon, draw-2D-quads, draw-2D-shape, draw-2D-text, draw-2D-texture, draw-2D-triangles, etc.

To be able to use these primitives in a meaningful way, some knowledge about the musical context of the note head is needed. Access to this information is provided through some special variables and objects which are enumerated in the following list:

(1) `x` and `y`. These variables give the corresponding position of the note head in the horizontal and vertical axis (see Figure 5).

(2) `notehead`. A character or string defined by the user.

(3) `width`. This variable represents the length or extent of the notehead. As can be seen in Figure 5 this is mainly determined automatically according to the duration of the associated note. It can also be edited by the user through the GUI or algorithmically.

(4) `height`. Some note heads (e.g., bpf note head) provide information as to their height. For the others this attribute is set to 1.0. This parameter is usually user-editable through the GUI.

(5) `stem-down-p` flag is true if the chord containing the note has the stem downward.

(6) `pitch` gives the midi value of the note. It can also contain a fractional part in case of micro-intervals.

(7) `augmentation`. Optional augmentation (e.g. dots). These are calculated automatically by ENP according to the duration/rhythm of the note.

(8) `self`. This variable gives the actual note object. There are several attributes (in addition to the ones given above) that can be read directly from it. In most cases it is not required to access any information in this way. It is nevertheless provided for advanced applications.

Figures 6 and 7 illustrate the use of the `width` parameter. The two lines of code in the code view both draw a single line segment. The first line segment is drawn from $\{0.0, 0.0\}$ to a fixed end point at $\{2.0, 2.0\}$. The end points

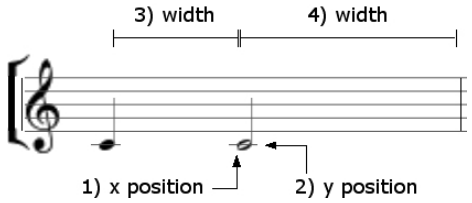

Figure 5. The attributes of ENP note heads: (1) x position, (2) y position, and the widths of two notes of different lengths (3) and (4).

of this line are always drawn relative to the position of the note head. The second line of code, on the other hand, draws a line segment from $\{0.0, 0.0\}$ to a variable end point at $\{\text{width}, 2.0\}$. The behavior of these two line segments can be observed by comparing the preview scores found in Figures 6 and 7. In the latter score the measure is considerably wider than in the former one, thus giving more room to the note. Notice, how the shorter line segment preserves its length and position but the longer one adjusts itself according to the new width of the note.

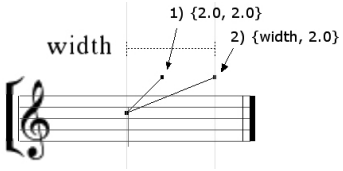

Figure 6. Two line segments drawn in the ND canvas. Both have one fixed end point at $\{0.0, 0.0\}$. The former one (1) has a fixed end point at $\{2.0, 2.0\}$ and the latter one (2) has a variable end point at $\{\text{width}, 2.0\}$.

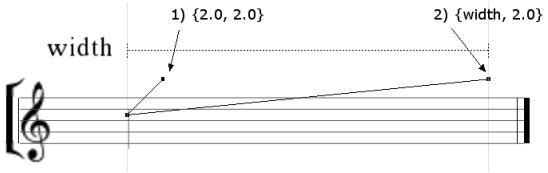

Figure 7. The variable end point $\{\text{width}, 2.0\}$ of the longer line segment dynamically follows the width of the note head.

3.3. Graphical Tools

In ND there is also a collection of graphical tools that can be used to compose the visual appearance of the note heads. The currently available set of tools consist of the following:

(1) Line segments, line strips, arrows, filled or framed polygons, and circles. These tools are quite self explanatory. In general they behave as the ones found in any graphically oriented software. There are, however, some differences. In case of most of the primitives the user can, for example, add and delete points (or vertices) to form more or less complex shapes.

(2) Text. This tool allows to add textual information. The user can select the color, transparency, size, and font face. The currently available font faces are Times and Sonata.

(3) Line thickness, and stipple can be set to most objects. There is a set of predefined stipple patterns and the user can also enter any arbitrary pattern (a 16-bit pattern determines which fragments of a line will be drawn).

(4) Color and transparency. Color and transparency values can be chosen from a palette of some predefined colors and levels of transparencies. Additionally, any color provided by the system color chooser dialog can be used.

(5) Textures can be applied to polygons. Currently the texture coordinates automatically map to the vertex coordinates of the polygon (see [12] for further reference on texture coordinates).

In the following (see Figures 8 and 9) we give an example how the ND graphics canvas behaves. The note head appearance is always drawn relative to the origo, i.e., the x/y coordinate of the note head. If the note head of the chord containing the note head is transposed or otherwise displaced in the score the graphical objects follow accordingly. This allows to check the behavior of the note head in several different positions.

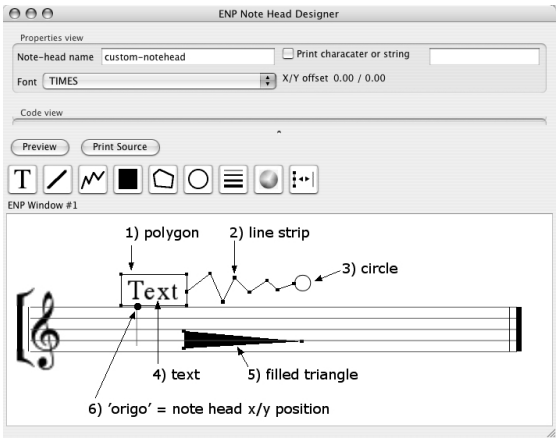

Figure 8. The ND graphics canvas shows some graphical objects with the editable handles shown: (1) polygon, (2) line strip, (3) circle, (4) text, and (5) filled triangle. The origo of the drawing canvas is given in (6). All the graphical objects are drawn relative to this point.

4. SOME NOTE HEAD DESIGNER EXAMPLES

In this section we use ND to design several note heads. We begin by defining a simple character-based note head. Then we move to more complex cases as we define three hand drawn note heads. The first two are static ones, meaning that they retain their shape regardless of their musical context. The third one, in turn, is a dynamic one as it adjusts its visual appearance according to the extent of the note. As the final example we define an algorithmic note head that graphically visualizes the production of a rattling sound.

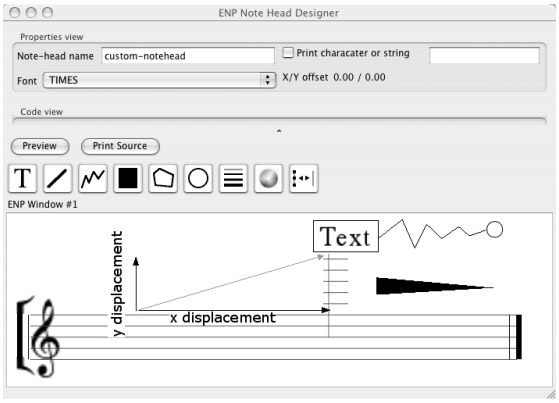

Figure 9. The chord (containing the designed note head) has been manually displaced both in x and y dimensions (transposed up and dragged to the right). The graphical objects adjust their position accordingly.

4.1. A Character-based Note Head

This approach is the most simple one. The user can define a character found in either Times or Sonata font faces to be used as the note head. In this example we have selected a '©' symbol to create a note head that could potentially be used, for example, to represent a pitch-wise constrained note (see [6]). Figure 10 gives the corresponding ND window. Although the definition of the note head itself seems trivial there are nevertheless several things that are handled on the behalf the user. Among other things the drawing method is defined along with the needed menu components.

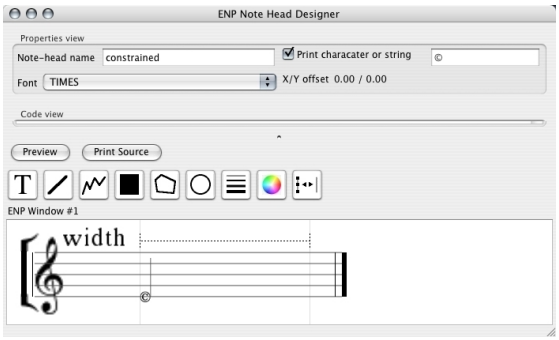

Figure 10. A simple character based note head created with ND.

4.2. A Hand-Drawn Note Head

Using the set of graphical tools described in section 3.3 makes it possible to draw the note head shapes as they

would appear in the score.

As our first hand drawn example we define a simple note head composed out of some characters and an arrow (Figure 11). This particular note head means scraping along the stings with the fingernail [10]. The arrow indicates the direction. In addition to the actual appearance of the note head the user needs to define a unique name ('scrape').

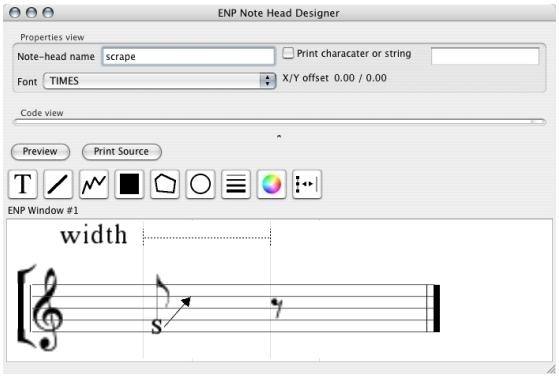

Figure 11. A note head with a character component and some simple graphics.

Next we use a filled polygon tool to draw a triangle shaped note head (Figure 12). The default polygon object contains four vertices so one vertex is removed simply by selecting and deleting it. One of the remaining vertices can now be dragged to form a triangle shape. Our new note head denotes an inhaling sound as used by, for example, Ligeti in 'Aventures'.

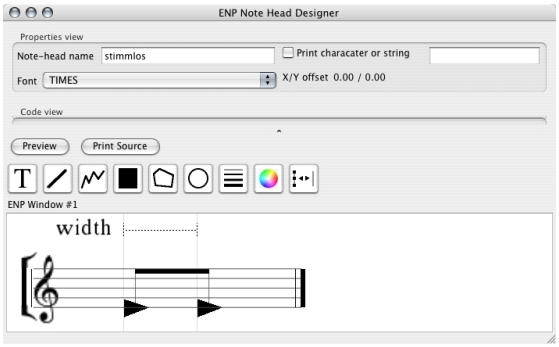

Figure 12. A static graphical note head. As the preview score is fully editable the second note head has been inserted in the score by copying.

The latter 1/8 note, in this example, was inserted in the preview score to illustrate the fact that the note heads are ready to be used in the current ENP session. They behave exactly like the built-in ones and can be copied, inserted, and edited even in the preview score.

The two previous examples have dealt with relatively static note heads. However, it is sometimes convenient that the note head scales itself according to the space reserved for it in the score. In ENP the note heads can be made to fill the space between two notes or to justify itself to some proportion of it. The Figure 13 shows an example of an object of this kind. The note head could be interpreted to denote, for example, an evolving cluster (e.g., à la Penderecki).

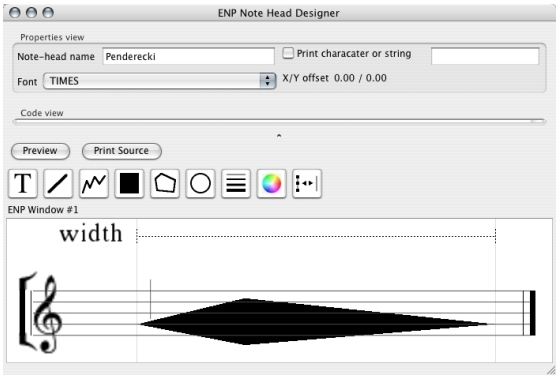

Figure 13. A dynamic graphical note head containing some proportionally spaced vertices.

This note head is drawn using the filled polygon tool. The polygons, in turn, are constructed out of a set of vertices. A vertex can have two different modes: a normal mode and a proportional mode. Vertices that are in a normal mode have absolute x positions in the coordinate space, i.e., a vertex placed in $\{1.0, 1.0\}$ remains in that position unless moved explicitly by the user. The proportional vertices, however, calculate their x position according to the space given to the note (`width`). Note, that the y position of both kinds of vertices is always calculated relative to the y position (read: pitch) of the note, as explained above. Figure 14 gives a closer view of this particular example with the proportional vertices shown.

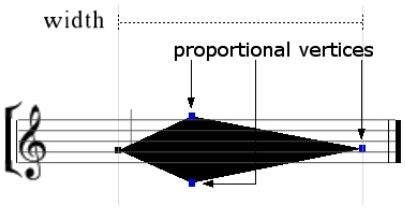

Figure 14. A note head drawn using a filled polygon tool. The figure shows also some proportionally spaced vertices.

4.3. An Code Based Note Head

As our last example we use the code view to define a Lisp-based note head. We implement a note head for a 'rattling' effect as described for example in [2]. The code is used to join a series of Bézier curves to form a complex, evolving curve that completely fills the space reserved for the note head.

Let us examine the source code in more detail (see code view in Figure 15). First, we define the outer contour of the shape and define a scaling factor for each of

the Bézier curves (`y-shape, y-scaling`). We basically draw the same curve repeatedly in a loop each time with a different transformation and scaling values (see the two OpenGL macros `with-GL-translate` and `with-GL-scale` in the bottom of the code view). Every other curve is also flipped vertically, using a negative scaling value, to smoothly join the end of each curve with the beginning of the next one. The whole shape is scaled inside the extent of the note by dividing the width of the note by the number of individual curves in the note head shape (the length of the `y-shape`).

The preview score in Figure 15 shows the final note head shape.

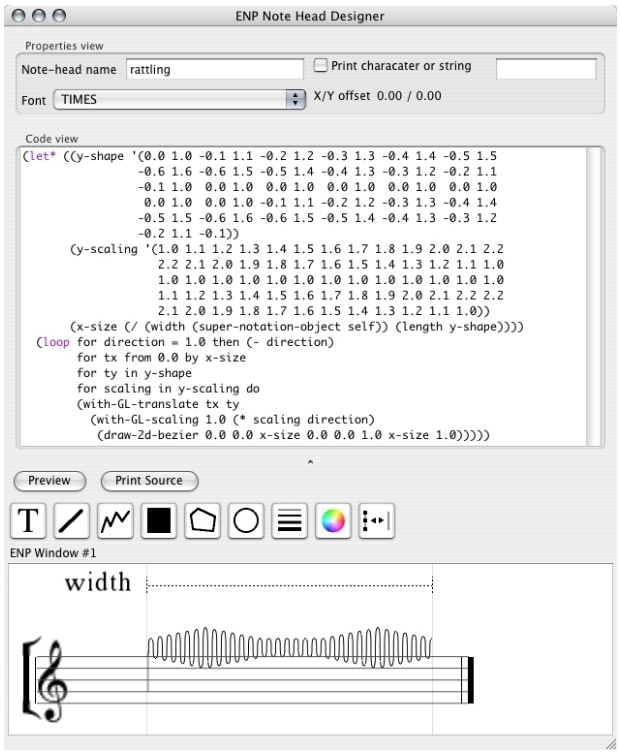

Figure 15. A complex note head shape composed out of several Bézier curves.

5. CONCLUSIONS

A new visual tool called ENP Note Head Designer was introduced along with several workable examples. The general note head scheme of ENP was also covered. The visual interface described in this paper allows to define note heads using either a text-based or graphically oriented approach. This allows users with different backgrounds in programming to take advantage of this tool. Furthermore, the uniform representation of note heads in ENP allows several advantages. First, it is possible to use the new note heads in the current ENP session without any compiling or reloading phase. Second, it is also possible to use the new note heads along with ENP-score-notation, the text based format of ENP (see [5, 7]). There is no need for any additional coding; the note heads behave as any of the build-in ones.

Finally, there are some improvements planned in ND:

(1) Usually the note heads may have slightly different parameters and offsets depending whether they are printed stems up or stems down. The ND window could potentially be organized so that there are two different detail views (one for up stem and another for down stem) and one overview which would display both versions at once. Currently it is not possible to distinguish between up or down stem variants other than using the code based approach.

(2) There should also be a possibility to group the note head entries in the menus, much in the same way as expressions are currently grouped [4]. This would make it more convenient to write dedicated libraries for specialized cases such as modern notation or shape-notation.

(3) There are still some useful tools missing. (a) Grouping/Un-grouping. With this tool the user could group several objects to form a unit that could then be moved simultaneously. (b) Rotation. All objects should be allowed to be rotated in a 2D space either interactively or additionally by providing a rotation angle.

(4) There should be a collection of more dedicated graphical tools such as triangles, squares, etc. This would allow more specialized behavior, thus, when zoomed, a square would retain its shape and so on. Now all the points forming a polygon can be moved individually.

(5) A grid with snap-to-grid option should probably be offered. There should be the possibility to choose individual vertical and horizontal grid spacings (e.g., proportions of the width horizontally, and fractions of the line spacing or staff height vertically).

(6) Finally,there should also be a possibility to save the current ND session. This would allow to recall a note head in an editable form and continue to work with it. Now, only the note head definitions can be saved in a file.

6. ACKNOWLEDGMENTS

The work of Mikael Laurson has been supported by the Academy of Finland (SA 105557).

7. REFERENCES

[1] MakeMusic! Inc. *Finale User Manual*.

[2] Erhard Karkoschka. *Das Schriftbild der Neuen Musik*. Hermann Moeck Verlag, Celle, 1966.

[3] Mika Kuuskankare and Mikael Laurson. ENP2.0 A Music Notation Program Implemented in Common Lisp and OpenGL. In *Proceedings of International Computer Music Conference*, pages 463–466, Gothenburg, Sweden, September 2002.

[4] Mika Kuuskankare and Mikael Laurson. ENP-Expressions, Score-BPF as a Case Study. In *Proceedings of International Computer Music Conference*, pages 103–106, Singapore, 2003.

[5] Mika Kuuskankare and Mikael Laurson. Recent Developments in ENP-score-notation. In *Sound and Music Computing '04*, October 2004.

[6] Mikael Laurson. Recent Developments in PatchWork: PWConstraints - a Rule Based Approach to Complex Musical Problems. In *Symposium on Systems Research in the Arts*, Baden-Baden, 1999.

[7] Mikael Laurson and Mika Kuuskankare. From RTM-notation to ENP-score-notation. In *Journées d'Informatique Musicale*, Montbéliard, France, 2003.

[8] Sibelius Software Ltd. *Sibelius3 User Guide*.

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

[10] Howard A. Risatti. *New Music Vocabulary. A Guide to Notational Signs for Contemporary Music*. Univ. of Illinois Press, Urbana, 1973.

[11] Bill Schottstaedt. Common Music Notation. In *Beyond MIDI, The Handbook of Musical Codes*. MIT Press, Cambridge, Massachusetts, 1997.

[12] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide*. Addison Wesley, Massachusetts, USA, 3rd edition, 1999.