ENP EXPRESSION DESIGNER

A VISUAL TOOL FOR CREATING ENP-EXPRESSIONS

Mika Kuuskankare
Sibelius Academy
DocMus
mkuuskan@siba.fi

Mikael Laurson
Sibelius Academy
CMT
laurson@siba.fi

ABSTRACT

In the present paper we introduce a new tool called ENP Expression Designer (ED). It is aimed at visually creating new notational attributes (ENP-expressions) in ENP. ENP, in turn, is a music notation program written in Lisp and CLOS with a special focus on compositional and music analytical applications. Currently, ENP allows to create new expressions by using Lisp and CLOS. The strictly textual interface can, however, be demanding to those users that have limited or no experience in programming. ED provides an easy and straightforward way to define new expressions through a graphical user interface.

Through inheritance the ED allows to reuse the built-in properties of the existing ENP-expressions. We also provide a set of graphical primitives for drawing lines, text, etc. Additionally, more experienced users can take full advantage of the OpenGL graphical language.

1 Introduction

Especially in modern music notation there is the need for special expression markings. There is an increasing amount of instrument specific playing techniques and new ones are invented all the time. These markings can also be invented even piece per piece basis. This makes it difficult for a music notation program to provide a comprehensive collection of expression markings. Thus, it is more reasonable to provide an interface for the user that (s)he can use to build custom expressions.

Two of the most widely used commercial notation programs, Finale [1] and Sibelius [2], provide the possibility to create user defined expressions. In Finale it is possible to create either text expressions or graphic expressions (shapes in Finale terminology). In Sibelius, in turn, the former group of expressions is called Symbols and the latter one Lines. These programs provide tools to create both types of expressions. However, when using these tools it is usually not allowed to view the expressions in their natural context—music notation. In CMN [6] it is also possible to define new expressions using Lisp.

ENP [3] is a music notation program that has been developed in order to meet the requirements of computer aided composition, music analysis and virtual instrument control. It's purpose is to display scores using the common western notation. Furthermore, ENP provides a rich set of both standard and user definable notational attributes, called ENP-expressions. ENP is programmed in Common Lisp (LispWorks) and uses the OpenGL API to produce the notational graphics.

ED is a visual tool that can be used to create custom expressions in ENP. ED provides a simple protocol that helps to define the basic properties of the user definable expressions. It also allows to define the visual appearance using Lisp. There are several advantages in the approach presented in this paper. One advantage that ENP provides is that the defined expressions can contain user definable data. This means that the data can be read and written both through the graphical user interface or algorithmically. The data can then be used in, for example, compositional or music analytical applications. Second, the expressions can be dynamic since they are defined using standard Common Lisp. Third, in ED the expressions are displayed in a fully functional ENP score, thus the user can observe how the expressions behave, for example, when the music is transposed or when the layout of the score is changed. This approach is different form any of the existing ones. For example, in Finale, the user definable expressions (shapes) are designed without the presence of any musical context.

The rest of the paper is structured as follows: first, we give a brief overview of ENP-expressions. Next, we introduce the ENP Expression Designer and give a few practical examples of its use. We end the paper with some conclusions and suggestions for future work.

2 Overview of ENP-Expressions

ENP-Expressions are divided into two main categories: (1) standard expressions, including articulations and dynamics, and (2) non-standard expressions which include, for example, groups and Score-BPF's [3]. Every expression is attached either to a single notational object (*single expressions*) or to a group of notational objects (*group expressions*).

ENP provides a collection of standard tempo-, dynamics- and articulation-symbols. Furthermore, it includes a set of non-standard notational attributes. New expressions can be created through inheritance.

Figure 1 shows a collection of some standard and non-standard ENP-expressions. Next, we present some of the basic components of ENP-Expressions.

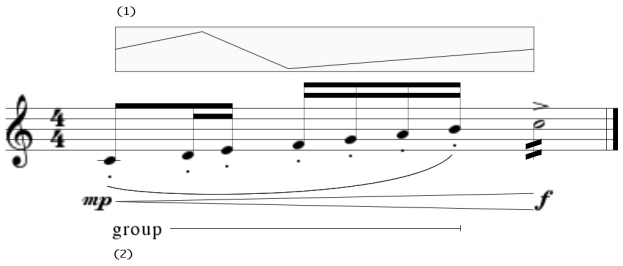

Figure 1. Some standard and non-standard ENP-expressions, including (1) score-BPF, (2) group.

2.1 The Basic Components of ENP-Expressions

All ENP-expressions share some common properties. These include, for example, print symbol, default print position, user definable offset, and font scaler.

While this is not a complete list these properties already define a bulk of the visual appearance, placement, and behavior of the expressions. The specific functions of each of the properties are further discussed in subsections 3.2 and 3.3. Figure 2 shows these properties in a musical context.

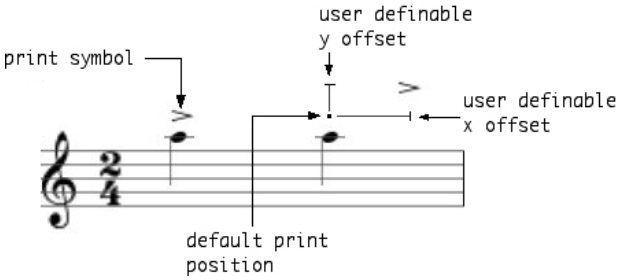

Figure 2. The components of an ENP-expression. On the left the graphical representation of an accent. On the right, some common attributes to all ENP-expressions.

3 ENP Expression Designer

ED is a visual tool for rapid expression designing and prototyping. The work-flow typically consists of three steps: (1) defining class and inheritance, (2) defining properties, and (3) defining the visual appearance (coding). All these steps can be executed in a single ED window. The ED Window, in turn, contains the following components: class view, properties view, code view, and preview score (an example ED window is shown in Figure 4).

Expressions are defined by first entering a unique class name and its superclass. Next, some additional properties are defined. These include among other the font typeface. Finally, the graphical outlook of the expression is defined by entering some Lisp code in the *code view*. The visual appearance of the new expression is shown in the *preview score* at the bottom of the ED Window.

There are also buttons for displaying the preview and generating the expression definition in Lisp. The *preview* -button refreshes the *preview score* to give the updated visual representation of the expression. The *print source* -button, in turn, generates a text document containing the Lisp code needed to store and restore the user defined expression. Change in any of the aforementioned properties provides synchronized feedback in the preview score.

In the following Subsections we examine in detail the components of the ED Window.

3.1 Class view

In the *class view* the user must specify a unique class name and a superclass. By default all the user defined expressions inherit from the built-in `expression` -class. This is the fundamental class that provides the necessary data structures and behavior of all ENP-expression.

3.2 Properties view

The *properties view* provides the following options:

(1) *Menu-category*. All user-defined expressions are automatically added in the appropriate ENP menus where they are, in turn, arranged in groups according to the type and/or purpose. Here the user can either choose one of the existing menu categories or create a new one by providing a new category name.

(2) *Print symbol* defines the string or the character which is used as the printed representation of the expression.

(3) *Print position*. There are several predefined print positions which, in turn, are used to determine the default print position. The current choices are: `:on-stem`, `:above-staff`, `:below-staff`, `:above-chord`, and `:notehead`. The individual effects of these values can be seen in Figure 3.

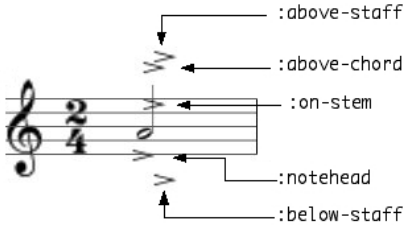

Figure 3. Five accent expressions attached to the chord. Each accent is given a different *print position*. The effect can be seen in the example along with the respective print position keyword.

(4) *Font* can currently be either `:times` or `:musical-symbols`.

There are also some check boxes in the properties view. These can be used to determine whether the expression is standard notation, and whether the expression is saved along with the score or not.

3.3 Code view

In the *code view* the user can write standard Lisp code (see Figure 4). The code defines how the expression

is drawn in the score. The users can utilize any of the OpenGL functions provided by LispWorks. There are also some special graphical primitives provided by ENP that can be used to draw, for example, lines, polygons, circles, etc. The following list gives the complete set of graphical primitives currently available: 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, and draw-2D-triangles. [1]

There are also some special variables that can be referred to in the code. The following variables contain some useful properties of the expressions:

(1) `x` and `y` give the absolute position of the expression. They are calculated according to the position of the associated notation object and other constraints. These values are an aggregate of the default position (calculated according to the print position) and the user definable offset discussed above (see Subsection 3.2).

(2) `width` gives the horizontal distance from the current notation object to the next one.

(3) `print-symbol` contains the string or character defined by the user in the *properties view*.

(4) `font` gives the font name as a keyword. Currently, the value of this variable is either `:times` (for readable text) or `:musical-symbols` (for music notation).

(5) `instrument` variable contains an ENP instrument object that, in turn, contains information about the name, range, etc.

(6) `expression` is the current ENP expression object that is being drawn. This object can also be queried about extra information such as any user definable data stored in the expression.

(7) `self`, in this case, refers to the notational object the expression is associated with (i.e., a note or a chord). Different properties of the notational object, including pitch, start-time and duration, can also be used in the expression code.

3.4 Preview score

The *preview score* displays the user defined expression as it would appear in a regular ENP score. The *preview score* is also fully editable so the user can observe how the expression behaves, for example, when the music is transposed, etc. The score can also be panned and zoomed freely so that the expressions can be examined and aligned in great detail.

There is one additional feature to this view. As described above, the default expression position is calculated according to the selected *print position* and an additional user definable offset. This offset can be adjusted in the *preview score* by dragging the expression with the mouse.

[1] The list of the graphical primitives is quite self explanatory and it is not further discussed in this paper.

4 Some Expression Designer Examples

In this Section we give two examples on how the ED can be used to create new expressions. In the first example, *wirebrush*, we build a simple percussion mallet symbol [5]. This example demonstrates the use of some of the graphical primitives. In the second example, in turn, we create an expression that uses the knowledge about the spacing of the associated notational objects to fill a given space in the score.

Due to space limitations the examples are given without any detailed explanation. The key points, however, are discussed along each example. Some knowledge about Lisp is beneficial to thoroughly understanding the provided code examples.

4.1 A Percussion Mallet Symbol: Wirebrush

As our first exercise we create a simple percussion mallet symbol denoting a wirebrush (see Figure 4).

We begin the expression definition by providing a unique class name, 'wirebrush' (1). In this case there is no need for any additional inheritance other than the basic expression behavior. Thus the default value ('expression') displayed in the superclass menu is sufficient for our purposes (2). The menu category is chosen to be 'User' (3). Print symbol and font typeface, in this example, do not have any use since the expression in question is purely graphical (4). In music notation these kinds of instructions are usually written above the staff [4] and the print position is selected accordingly (5). In code view there are three lines containing some drawing instructions (6). Finally, the graphical representation of the expression can be seen in the preview score (7). The second note in this example is added to demonstrate how the user-defined expression behaves when the associated notational object is transposed. Note, how the expression avoids colliding with music notation.

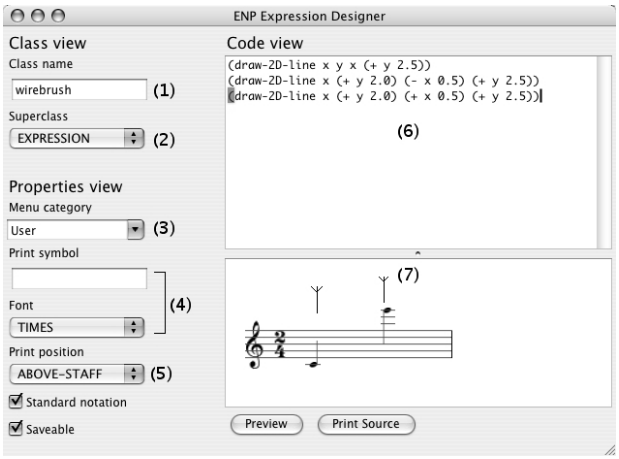

Figure 4. The wirebrush mallet symbol created with ED.

4.2 A Playing Technique: Scrape

The second ED example focuses on a graphic expression that can be used to denote scraping an instrument [5]. As we draw the scraping symbol in the score it is required have knowledge about the length or extent of the associated notational object (Figure 5). This information is provided with the special variable called `width`.

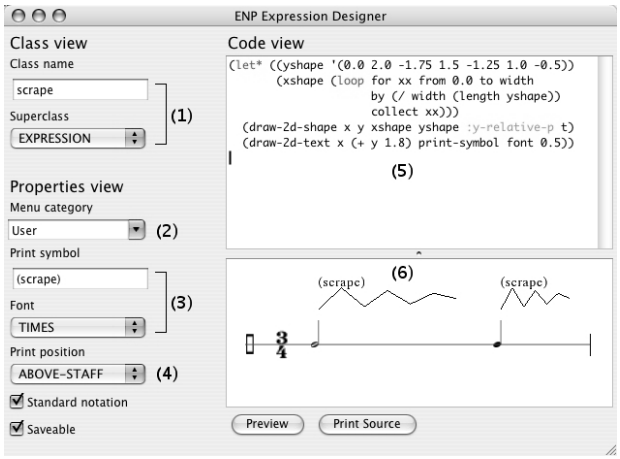

Figure 5. A playing technique expression created with ED.

Next, we briefly describe the steps needed to define the scraping expression. The class name and superclass are as shown in (1) and the menu category is again chosen to be 'User' (2). In this case, it is required to define the print symbol and font typeface since the expression contains some written instructions along with the graphical information (3). The expression print position is 'above-staff' as indicated in (4). The lisp code part is once again offered without any detailed explanation (5). Suffice to say that the x and y -shapes are first defined and then drawn with a graphics primitive called `draw-2d-shape`. Additionally, the user defined print symbol is also drawn. Finally, the preview score (6), demonstrates how this particular expression reacts to the duration of the associated notation object due to the `width` -parameter (see code view).

5 Conclusions and Future Work

The current version of ED is already quite functional and can be used to quickly prototype and create new expressions.

Some of the benefits in this approach are that it does not require extensive programming experience and the limited set of graphics primitives is quite easy to learn. It allows to create new expressions without the need to reproduce the low-level lisp code for each one. The number of lines of code the user is required to write is reduced to a fraction compared to the entirely text based approach. Furthermore, the synchronized visual feedback offered by ED is a valuable property as it allows the user to design and view the expressions in their native context.

There are, however, several additional features and enhancements planned to make ED more useful as a general purpose tool: First, it is currently not possible to create group expressions. Group expressions require more options and possibly a dynamic property view. There are also a handful of additional methods that are user definable which in turn require more code view type of inputs. A dedicated group expression tab would be one solution to lay out the ED Window to facilitate this feature. Second, it should also be possible to use graphical tools for drawing the expression definition. This feature would require a graphics canvas and a set of graphical tools, such as lines, polygons and circles. It would also bring the ED closer to the more traditional tools offered by, for example, Finale. However, allowing to combine both algorithmic and 'hand drawn' representations would make this approach more flexible. Third, there is currently no means to add slots to the user defined expression classes through ED. This is of primary importance when designing more complex objects that, for example, have different kinds of graphical representations depending on attributes selected by the user. Finally, although ED currently creates the menus for inserting the expression in the music notation there should be the possibility in ED to define the appropriate context sensitive menus for the expression itself. This would require a dedicated menu editor to be embedded in ED.

The ED project was launched only recently and as can be seen in the missing features list above there is still quite a lot of work to be done. The current version is more of a proof of concept at this time. It requires some redesigning and additional coding to integrate all the listed features in ED. However, the fully mature application should prove to be beneficial not only for casual users but also for the more experienced ones.

6 Acknowledgments

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

7 References

[1] *Finale User Manual*.

[2] *Sibelius3 User Guide*.

[3] Kuuskankare, M. and M. Laurson. "ENP2.0 A Music Notation Program Implemented in Common Lisp and OpenGL", *Proceedings of the International Computer Music Conference*, Gothenburg, Sweden, 2002.

[4] Read, G. *Music Notation*. Victor Gollancz Ltd., 1982.

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

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