From RTM-notation to ENP-score-notation

Mikael Laurson, Mika Kuuskankare

From RTM-notation to ENP-score-notation

Mikael Laurson[1] and Mika Kuuskankare[2]

[1]Center for Music and Technology,
[2]Department of Doctoral Studies in Musical Performance and Research.
Sibelius Academy, P.O.Box 86, 00251 Helsinki, Finland
laurson@siba.fi, mkuuskan@siba.fi

Abstract

This paper discusses some recent developments within a compositional environment called PWGL. Our focus is to present how score information is represented in PWGL. We give some background information concerning the rhythmic notation that was used in PatchWork (a predecessor of PWGL). After this we show how this notation has been expanded so that it allows to generate very detailed scores that can contain besides the basic rhythmic structures also other information such as grace-notes, instrumentation, pitch and expressions.

1 Introduction

PWGL (Laurson and Kuuskankare 2002) is a visual programming language based on Lisp, CLOS and OpenGL. PWGL is tool for computer assisted composition and sound synthesis. PWGL is based on similar concepts than PatchWork (PW, Laurson 1996) such as direct relation to its base languages Lisp and CLOS, musical objects and visual editors. The main difference between PW and PWGL is that the graphics part of the implementation is based now on OpenGL and not on QuickDraw as was the case with PW. OpenGL (Woo et al. 1999) has many attractive features and offers several advantages when compared to QuickDraw (for more details see Laurson and Kuuskankare 2002).

In this text we will concentrate on the way complex musical scores are represented in PWGL. Scores are of primary importance in our system and they can be used in many compositional and analytical applications such as to produce musical material for instrumental music. Scores have been also used successfully to produce control information for physical models of acoustical instruments (see for instance Laurson et al. 2001 and Välimäki et al. 2003).

The Score-editor of PWGL utilizes internally the ENP2.0 music notation package. ENP2.0 (Kuuskankare and Laurson 2002) is based on the former ENP user library of PW (Kuuskankare and Laurson 2001). The Score-editor combines all PW music notation related editors (i.e. chord-editor, chord-sequence-editor, RTM-editor, PolyRTM-editor) into one package. ENP2.0 supports both mensural and non-mensural notation. Also ENP2.0 has been completely rewritten and is now based on OpenGL. The purpose of ENP2.0 is to provide professional notational capabilities, a graphical user interface and powerful object structures for compositional use. As PWGL, ENP2.0 is programmed with LISP and CLOS thus being extendible and open. It also provides full access to the musical structures behind the notation allowing ENP2.0 to be controlled algorithmically. Furthermore, ENP2.0 provides both standard and user definable expressions.

Algorithmic control requires a high-level representation of the complex structural information behind the visual front-end of a musical score. To be useful this representation has to fulfil the following criterias. First, it has to be compact so that we are able to handle large scores efficiently. Second, it should be understandable to a human reader so that it could be used directly as input to build scores. Third, it should be easily translatable into object structures of the final score, and vice versa, any score should be translatable back to the high-level representation. Finally, it should be extendible in order to meet future improvements and extensions of the current environment.

The RTM-package (Laurson 1996) of PW allowed to generate complex rhythms out of a list representation for beats. Henceforth we will call this list representation *RTM-notation*. In the current PWGL implementation the RTM-notation representation has been augmented in several ways.

In the following we first introduce the PW RTM-notation and describe its most important features. After this we go over to the current implementation and show step by step how it can handle scores with increasing complexity. We discuss also one elaborate score example and end with some concluding remarks.

2 RTM-notation in PW

The section gives a synopsis of the RTM-notation syntax used in PW to represent scores. We start with Lisp lists that are converted to metrical rhythms. Lisp lists is a natural choice as they allow easily to build hierarchical structures needed for our purposes. In the following we show the basic components of the system and after this discuss several examples that can be found in the western notation practice (Read 1980).

2.1 Beat-list

The starting-point for a Lisp representation of a beat is a list. This list is called a *beat-list* and has two components: *beat-count* and *rtm-list*. Beat-count is a positive number, usually an integer. rtm-list, in turn, is a list of numbers. The numbers, called *rtm-values*, are usually positive integers, but can also be negative integers or floats. Let us assume the following example (Figure 1):

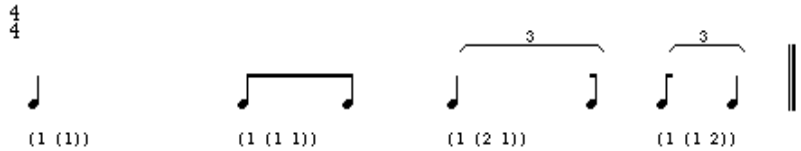

Figure 1: Some basic rhythms with their beat-list representations.

Figure 1 gives a measure containing four beats. Below each beat is the corresponding beat-list. Thus, for example, for the first quarter-note, the beat-list is (1 (1)). This detail is given also in Figure 2:

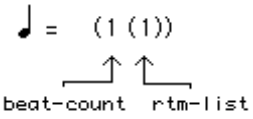

Figure 2: A quarter-note and its beat-list representation.

beat-count indicates how many units of the basic pulse is needed for a beat. The basic pulse, in turn, can be inferred from the denominator of the time signature. Thus, in the example above, the basic pulse is a quarter-note. As the beat-count is 1, the length of the beat in Figure 2 will be one quarter-note.

The rtm-list consists of proportional values and has three properties: (1) the sum of all numbers in the rtm-list, (2) the length of the rtm-list and (3) the values of the distinct numbers. The sum gives the number of units into which the beat must be divided. The sub-unit of the beat equals 1 divided by the sum. Next, the length of the rtm-list determines the number of notes inside a beat. Finally, the rtm-value *n* indicates that *n* sub-units are to be assigned to the corresponding note. As the rtm-list in our example consists of only the number 1, we have only one note with a length of a quarter-note.

Figure 3 gives the second beat of Figure 1, with the beat-list (1 (1 1)). The beat lasts for one quarter-note (beat-count = 1). As the sum of the rtm-list elements is 2, the sub-unit is 1/2. The beat contains two notes, both being 1/2 of the length of the beat, i.e. eighth-notes (1/4 * 1/2 = 1/8).

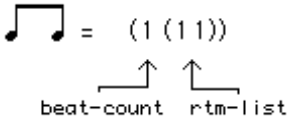

Figure 3: Two eighth-notes and the corresponding beat-list.

In the third beat of the example in Figure 1 above, the beat-list is (1 (2 1)). The beat will last for one quarter-note (beat-count = 1). As the sum of the rtm-list is 3, the sub-unit is 1/3 of the length of the beat. The length of the rtm-list is 2. The length of the first note is 2/3 (2 * 1/3) of the beat, that of the second note 1/3 (1 * 1/3). Thus, the first note is a "sixth-note" (1/4 * 2/3), the second a "twelfth-note" (1/4 * 1/3).

In the fourth beat of Figure 1 the rtm-list is (1 2), as opposed to the (2 1) of the previous beat. Consequently, the length of the first note will be 1/3 of the length of the beat, that of the second 2/3.

2.2 Simple Rhythms

The examples below (Figures 4 - 6) show beat-list representations describing rhythms with different sub-units. The sub-units range from 1/4 to 1/6:

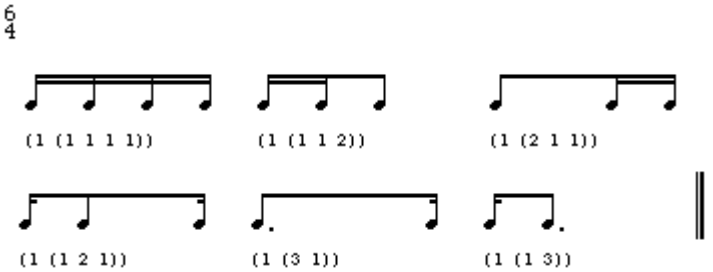

Figure 4: sub-unit = 1/4.

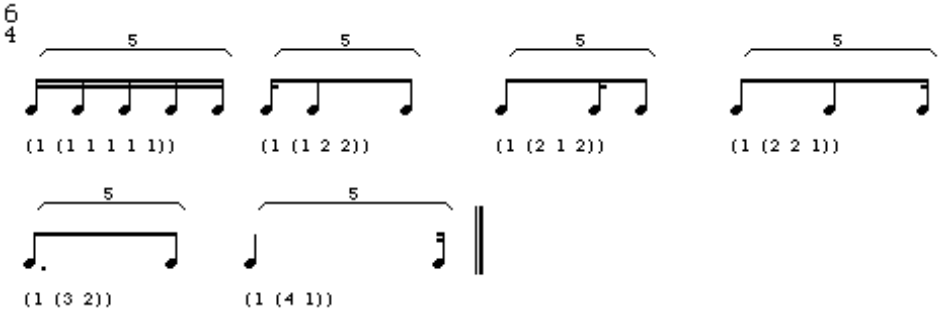

Figure 5: sub-unit = 1/5.

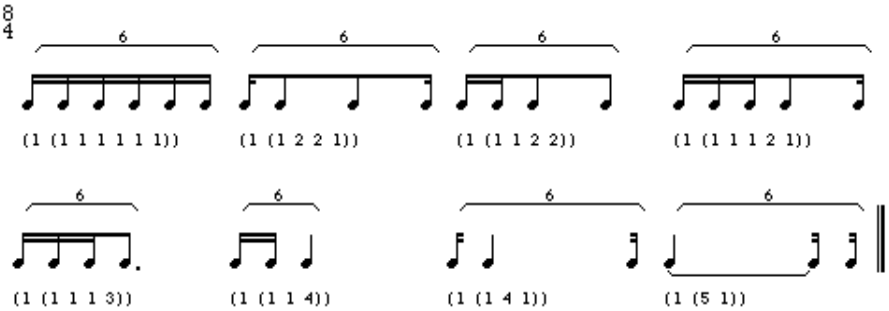

Figure 6: sub-unit = 1/6.

Figure 7 gives an example with beat-count 2. Hence, each beat gets two units (2 * quarter-note) of the basic pulse:

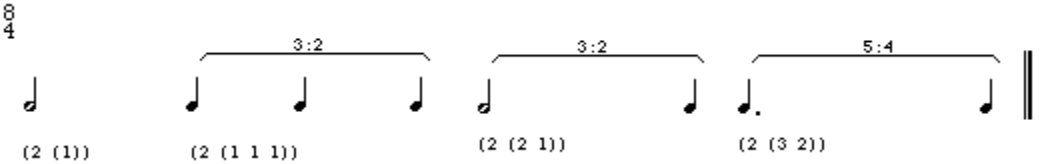

Figure 7: beat-count = 2.

2.3 Embedded Rhythms

Until now all the rtm-lists have been simple lists, that is, only one level deep. To be able to represent embedded rhythms, we add more levels to the rtm-list. In Figure 8, the top example starts with a simple beat-list (1 (1 1)). Then, in the right-hand side of the top example we replace the second 1 of the rtm-list with a beat-list (1 (1 1 1)). The result is a new embedded beat-list: (1 (1 (1 (1 1 1)))). The second example starts with a beat-list (2 (1 1 1)). We replace each 1 of the rtm-list with (1 (1 1 1 1 1)).

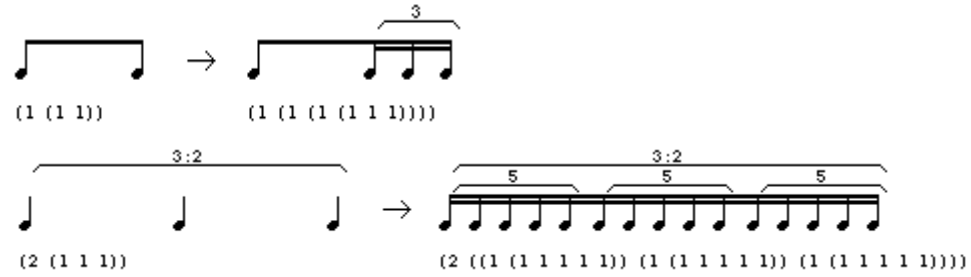

Figure 8: The construction of embedded rhythms.

2.4 Compound Rhythms

In *compound rhythms* the grouping of notes is controlled by the beat-count. In Figure 9 the time signature is 8/8 and the grouping is 3 + 3 + 2.

Figure 9: Compound rhythms.

2.5 Rests and Tied Notes

Until now, all the rtm-list examples contained only positive integers. To represent rests and tied notes, we allow the numbers inside the rtm-list to be negative numbers or floats. If a number in a rtm-list is negative, it is considered to be a rest. If it is a float, it represents a tied note. The rhythm in Figure 10 shows a combination of note-heads, rests and tied notes.

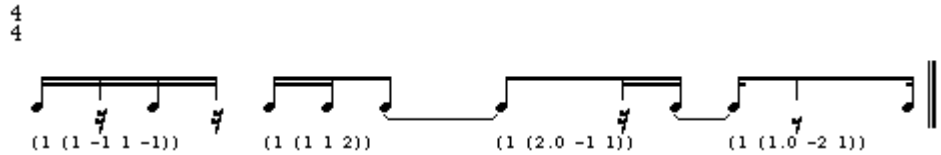

Figure 10: A combination of note-heads, rests and tied notes.

3 ENP-score-notation in PWGL

In this section we present a new and enriched list notation called *ENP-score-notation*. We describe how the ENP-score-notation concept differs from the RTM-notation presented above. ENP-score-notation is based on some important concepts of RTM-notation, such as the hierarchical PW beat-list. For several reasons, this approach was adapted also in ENP. First, we wanted to ensure basic backwards compatibility between PW music editors and those of PWGL. Second, the list representation is uniform and proven to be powerful in practical use. And third, it was easy to extend the expressiveness of the syntax, as required by ENP, without scarifying the functionality or backwards compatibility.

In ENP the musical structures of PW have been modified and extended to add new functionalities and hierarchies (Kuuskankare and Laurson 2001). Thus, also the ENP-score-notation is enriched to reflect these changes. The starting point in the current implementation is to use the PW beat-list representation. Using the notation given above it is possible to create complex beat structures. To create higher-level structures (i.e., measures, voices, parts, scores) we collect lower-level structures as lists. For instance, beat-lists can be used to build measures. The measures, in turn, can be used to build even higher-level structures.

3.1 ENP Score Structure

An ENP score is built out of hierarchical object structures. Thus a *score* consists of a list of parts, a *part* consists of a list of voices, a *voice* consists of a list of measures and finally a *measure* consists of a list of beats. For example we can create a score out of the following list representation that reflects directly the ENP score structure (note that both measures consists of beat-lists as described in Section 2).

```
(                                              ; score
  (                                            ; part1
    (                                          ; voice1
      ((1 (1)) (1 (1 1 1)) (1 (1 2 1)))        ; measure1
      ((1 (1 2 2)) (1 (1 2)) (2 (-1 2 1)))     ; measure2
    )
  )
)
```

Figure 11 shows in practice how the previous ENP-score-notation example is used in PWGL to build scores:

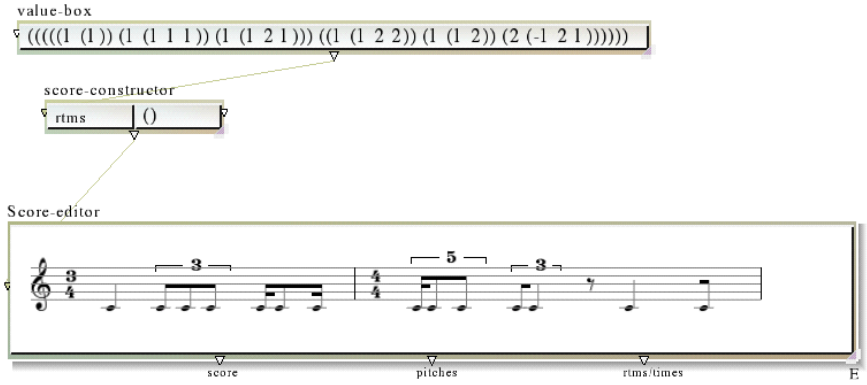

Figure 11: A PWGL patch constructing scores.

3.2 Keyword Extensions

The basic ENP-score-notation system described until now can be augmented by inserting keywords inside the list structure. This allows to define more precisely the resulting score. A typical example is for instance a measure that should have knowledge of its time signature in

order to work properly. If no information is given then default values are used. For instance in the example of Figure 11, the default denominator value of the time signature (or *:low*) is a quarter-note or 4 (the nominator value is calculated automatically using the beat-count values of the beat-lists), the default pitch value (or *:midi*) of a note is middle-C or 60, and so on. Typically the used keywords are slot-names of the resulting CLOS objects. This feature is useful as it allows to add new slots in the existing musical objects without having to modify the source code that takes care of the translation of the ENP-score-notation to an ENP score.

Each level of the ENP score structure has its own set of keywords. For instance, following keywords can be used: *:instrument, :staff*, *:clef-number, :low*, :*metronome-value*, *:start-time*, *:notes*, *:expressions*, *:midi, :velocity* and *:channel*.

Here are some ENP-score-notation examples dealing with various keywords accompanied with the resulting score (Figures 12 - 14):

```
((:instrument "Piano" :staff piano-staff
   ((:low 8 :metronome-value 72
      (1 (1 1 1)) (1 (1 (1 :notes ((60 :clef-number 0) (60 :clef-number 1)))))))))
```

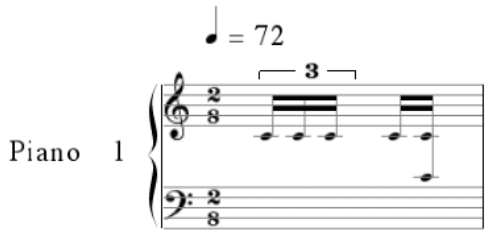

Figure 12: A score with keywords for instrument, staff, time signature, clef-number and metronome.

```
(((((4 ((1  :notes (72 64)) (1 :notes (72 65)) (1 :notes (72 67)) (1 :notes (71 65))))))
  (((4 (2 (2 :notes (62)))))))
```

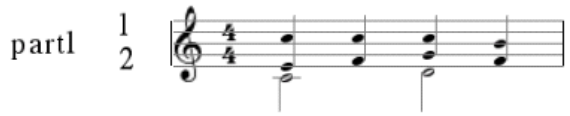

Figure 13: A two-voice part example with pitch information.

```
(:Title "Duo" :spacing 0.7
        (:instrument "Flute"
                     (((1 ((1 :notes (72))))
                       (1 ((1 :notes (72) :expressions (:staccato))
                           (1 :notes (74) :expressions (:staccato))
                           (1 :notes (76) :expressions (:staccato))
                           (1 :notes (77) :expressions (:staccato))
                           (1 :notes (78) :expressions (:staccato))) 'grace-beat)
                       (3 ((1 :notes (79) :expressions (:trillo)))))))
        (:instrument "Violoncello" :staff bass-staff
                     (((1 ((1 :notes (36 43) :expressions (:accent :f))))
                       (3 ((1 :notes (52 60) :expressions (:accent))))))))
```

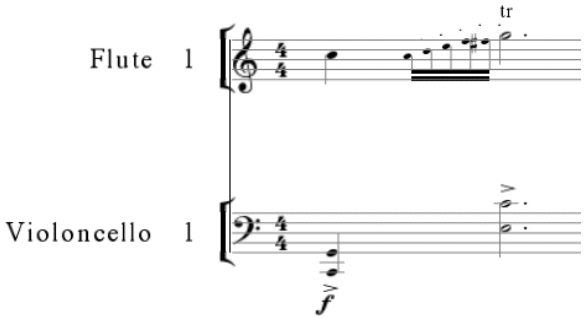

Figure 14: A two part example with instrument, staff, pitch and expression information.

3.3 Special Extensions

The PWGL ENP-score-notation system allows also to give extra information that defines the type of a beat-object. One example of these special structures is the *grace-beat*. Grace-beats are used to define grace-chords or grace-notes in an ENP score. An example of this can be found in Figure 14 where the upper part contains an ascending grace-note sequence. Grace-beats are given by inserting at the end of the beat-list definition the symbol *grace-beat*. The following expression:

```
(((((1 (1 1) 'grace-beat) (1 (1 1 1)) (1 ((1 (1 1)) (1 (1 1 1) 'grace-beat) (1 (1))))))))
```

gives as a result the score that can be seen in Figure 15:

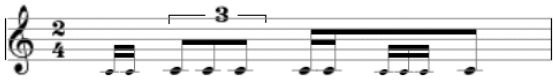

Figure 15: An example with two grace-beats.

A similar extension allows to create *accelerando-* or *ritardando-beats*. The following example shows how to define these special beats (see also Figure 16):

```
(:spacing 0.33 :zoom 2.0 :pan-y -55.0
  ((((1 (1 1 1 1 1 1 1) 'accelerando-beat)
     (1 (1 1 1 1 1 1 1) 'ritardando-beat)))))
```

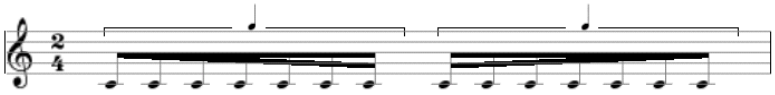

Figure 16: An example containing one accelerando-beat and one ritardando-beat.

3.4 Group-expressions

ENP expressions can be applied to a single note or chord or to a group of notes or chords. We have already seen several examples of the former expression type. This subsection discusses the latter case which is called a *group-expression*. Group-expressions can overlap freely in the score. They are interesting for our study as typically they do not obey the strict metrical hierarchy imposed by the beat-list and measure notation. Thus we need a special syntax to generate them in a metric context.

Group-expressions can be inserted to a chord using an appropriate keyword to define the class and instance identity of the expression in question. The keyword consists of two parts. It begins with an ENP class-name (e.g., 'group', 'slur', 'bpf', etc.). In addition of this, there can be an optional numeral that defines a group identity. The group identity numeral is used to distinguish between different instances of group-expressions. Thus, ':group1' would denote an

ENP object of class 'group' with identity value '1'. In this case, all subsequent references to ':group1' would point to the same instance.

Furthermore, when an expression is introduced for the first time, any number of initialization-keywords can be supplied. In this case, the expression is given as a list consisting of the expression keyword and the required initialization-keyword/value pairs, for example: (:group1 :print-symbol "3-11a"). Below we find an example with three group-expressions:

```
(((((1 ((1 :notes (73) :expressions (:slur1))
        (1 :notes (75) :expressions (:slur1 (:group1 :print-symbol "M1") ))
        (1 :notes (76) :expressions (:slur1 :group1))
        (1 :notes (78) :expressions (:slur1 :group1 (:group2 :print-symbol "M2")))))
    (1 ((1 :notes (74) :expressions (:slur1 :group1 :group2))
        (1 :notes (76) :expressions (:slur1 :group1 :group2))
        (1 :notes (77) :expressions (:slur1 :group1 :group2))
        (1 :notes (79) :expressions (:slur1 :group1 :group2))))
    (1 ((1 :notes ((75 :enharmonic 1)) :expressions (:slur1 :group1 :group2))
        (1 :notes (77) :expressions (:slur1 :group2))
        (1 :notes (79) :expressions (:slur1 :group2))
        (1 :notes ((80 :enharmonic 1)) :expressions (:slur1 :group2)))))))
```

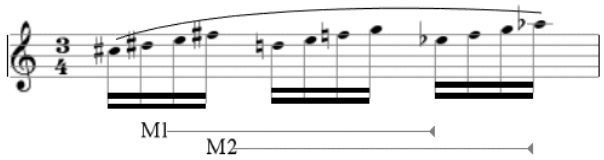

Figure 17: Group-expressions example containing a slur and two overlapping groups called 'M1' and' M2'.

4 Score example

We end with a complex ENP-score-notation example which combines several features described in the previous subsections. The resulting score is found in Figure 18.

```
(((((2 ((1 :notes (52) :expressions ((:slur1 :slope -6.0) :pp))))
    (1 ((1.0 :notes (52) :expressions (:slur1))
        (1 :notes ((58 :enharmonic 1)) :expressions (:slur1))
        (1 :notes (59) :expressions (:slur1))
        (1 :notes (62) :expressions (:slur1))))
    (1 ((1 :notes ((63 :enharmonic 1)) :expressions (:slur1 :crescendo1))
        (1 :notes (65) :expressions (:crescendo1))
        (1 :notes ((66 :enharmonic 1)) :expressions (:crescendo1))
        (1 :notes ((68 :enharmonic 1)) :expressions (:crescendo1))
        (1 :notes (69) :expressions (:crescendo1))))
    (1 ((1 :notes ((72)) :expressions (:slur2 :f))
        (1 :notes ((73 :enharmonic 1)) :expressions (:slur2) :x-offset 1.0)
        (6 :notes (79) :expressions (:fp :crescendo2) :x-offset 2.0))))
   ((1 ((1.0 :notes (79) :expressions (:crescendo2))
        (-1 :notes (72) :expressions (:fz))))
    (2 (-1 (1 :notes ((72 :note-head :x)) :expressions
             (:accent-grave :ff (:group1 :print-symbol "slap" :expression-fore-color 0)))
           (1 :notes ((69 :note-head :x)) :expressions (:accent-grave :group1))
           (1 :notes ((73 :note-head :x)) :expressions (:accent-grave :group1))
           (1 :notes ((68 :note-head :x)) :expressions (:accent-grave :group1))
           (1 :notes ((66 :note-head :x)) :expressions (:accent-grave :group1))))
    (1 ((1 :notes (77) :expressions (:glissando1 :fp :crescendo3))
        (1 :notes ((80 :note-head :invisible)) :x-offset 1.0)))
    (1 ((1 :notes (83) :expressions (:glissando1 :fz :staccato :crescendo3)))))))
```

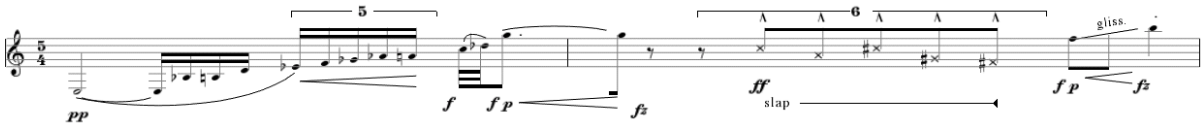

Figure 18: A complex example containing dynamics, single expressions, group-expressions and special note-heads.

5 Conclusions

This paper gave an overview of some recent PWGL developments dealing with a novel score representation called ENP-score-notation. We first introduced the PW beat-list protocol which is used to construct metric structures. After this we augmented the beat-list scheme with keywords and special extensions in order to be able to build scores containing parts, voices, measures, chords and notes. The system allows also to define additional notational attributes such as expressions, grace-notes and group-expressions.

References

Kuuskankare M. and M. Laurson. 2001. "ENP, Musical Notation Library based on Common Lisp and CLOS". In Proc. of ICMC'01, Havana, Cuba, pp. 131-134.

Kuuskankare M. and M. Laurson. 2002. "ENP2.0 A Music Notation Program Implemented in Common Lisp and OpenGL". In Proc. of ICMC'02, Gothenburg, Sweden, pp. 463-466.

Laurson M. and M. Kuuskankare. 2002. "PWGL: A Novel Visual Language based on Common Lisp, CLOS and OpenGL". In Proc. of ICMC'02, Gothenburg, Sweden, pp. 142-145.

Laurson M., C. Erkut, V. Välimäki, and M. Kuuskankare. 2001. "Methods for Modeling Realistic Playing in Acoustic Guitar Synthesis." Computer Music Journal 25(3).

Laurson, M. 1996. *PATCHWORK: A Visual Programming Language and Some Musical Applications*. Doctoral dissertation, Sibelius Academy, Helsinki, Finland.

Read, G. 1980. *Modern Rhythmic Notation. London*. Victor Gollanz Ltd.

Woo M., J. Neider, T. Davis, and D. Shreiner. 1999. *OpenGL Programming Guide*. Addison Wesley, 3rd edition, Massachusetts, USA.

Välimäki V., Laurson M., and C. Erkut. 2003. "Commuted Waveguide Synthesis of the Clavichord." To be published in Computer Music Journal.