INTRODUCTION TO COMPUTER-ASSITED MUSIC ANALYSIS IN PWGL

Mikael Laurson
Sibelius Academy, CMT
laurson@siba.fi

Mika Kuuskankare
Sibelius Academy, DocMus
mkuuskan@siba.fi

Kimmo Kuitunen
kpkuitunen@hotmail.com

ABSTRACT

We present in this paper our recent developments dealing with computer-assisted music analysis. Our focus is in a new syntax that extends the pattern-matching part of our constraint-based system called PWConstraints. PWConstraints can be used both to generate musical material or to analyze existing scores. The syntax allows to refer to more high-level entities in a score than before, resulting in compact analysis rules that use only a minimal set of primitives. Rules can return expression objects which can be used to visualize analytical information directly in a score. The compiler can be extended to support new score accessor keywords by special compiler methods. The new syntax and visualization of analysis data is explained and demonstrated with the help of several analysis examples.

1. INTRODUCTION

During the last 10 years the use of constraint-based languages in computer-assisted composition environments has found increased interest. Besides our constraint-based system, called *PWConstraints* [1], there are currently several other approaches oriented towards musical search problems such as Situation [2], Arno [3], OMClouds [4] and the more recent system by Anders based on the OZ programming language [5]. PWConstraints is written in Common Lisp and CLOS and is currently an integral part of our visual programming environment called PWGL [6]. When using our system as a search tool we do not formulate stepwise algorithms, but define a *search-space* and produce systematically potential results from it. Typically we are not interested in all possible results, but constrain these with the help of *rules* describing an acceptable solution. If the search cannot find an acceptable solution during the search process, then the system will *backtrack* in the search-space. This scheme thus allows to undo already accepted values and retry to satisfy the rules with a new set of values.

Although PWConstraints was originally designed to work as a tool to generate musical material it can also be used for music analysis purposes. In this case no backtrack search is involved and the system simply traverses through score and applies all analytical rules to the input score. Analytical rules have very similar syntax than the ones used in search problems. This kind of symmetry where a rule can be used either in a generative or analytical context is one of the main corner stones of our system. When a rule can be applied in both approaches the user can test and verify his/her rules in a more flexible way than using the generative approach only.

There are two main options in our system that can be used to realize analytical tasks. In the first one the user can choose a set of rules in conjunction with an input score and check whether the given rules are applied systematically in the input score. If a rule fails, information about the rule and the position of the failure will be printed in the output browser. In the second option—which will be the focus of this article—the rules can be used to add visual analysis information to various musical structures of an input score. The main difference between these approaches is that a rule in the latter case returns—instead of a simple truth value—a special expression object. This expression can be attached either to single score objects or to a group of objects. This mechanism has been described previously in [7].

Perhaps the most important system that is able to perform similar analytical tasks than PWConstraints is the Humdrum system by David Huron [8]. Humdrum allows researchers to encode, manipulate, and output a wide variety of musically-pertinent representations. The emphasis is on posing and answering questions about music. The system is textual based and requires the user to use Unix-kind command-line syntax. The *Humdrum Syntax* is a grammar for representing sequential symbolic information using ASCII (text) data. The *Humdrum Toolkit*, in turn, is a set of more than 60 inter-related software tools. These general-purpose tools manipulate ASCII (text) data that conforms to the Humdrum Syntax. Despite of some similarities our system is quite different from the Humdrum system as in PWConstraints the user can 'reuse' rules both in generative and analytical contexts. Humdrum, by contrast, is not well suited to generate musical data. Furthermore our system is strongly visual. For instance all score information can be manipulated by the user through a visual user-interface.

This paper describes a new syntax to write analysis rules that are able to visualize analysis information. For this purpose we need flexible musical data structures. Our starting point is our music notation package, called Expressive Notation Package (ENP, [9]), which allows to access all structural components of a score. Each rule should be able to dynamically extract required information out of a rich data structure, allowing the

rule to analyse the result from its own point of view. One of the main problems in formulating such rules is to find a clear formalism with which to point to the required data objects. This article presents a new approach that allows to access in a uniform way various structural entities from a score, such as chords, beats, measures and harmonic formations. A similar approach but in a generative context has been published by us in [10].

In the following we first introduce the general rule syntax in PWConstraints. We discuss some problems found in the previous implementation and propose a new syntax that allows to add in the rules score accessor keywords (Section 3). Next (Section 4) we give an overview how the user can define various analytical expression objects that will be added to the score during the analysis process. The new syntax is utilized to write melodic, harmonic and voice-leading rules (Sections 5, 6 and 7). We end with a larger example that aims to demonstrate how the system can be used in a concrete analytical context (Section 8).

2. RULE SYNTAX

In a PWConstraints analysis rule a pattern-matching language is used to extract relevant information from a score. This information is given to a Lisp function, called *Lisp-code part*, that either returns an expression object or nil. In the latter case the rule has no side effects and the score will not be changed.

The *pattern-matching part* of a rule uses a fairly typical pattern-matching syntax. It can contain *variables* (symbols starting with a '?'), *anonymous-variables* (plain '?'s), a *wild card* ('*') and *index-variables* (symbols consisting of an 'i' and an index number). A variable extracts single values. By contrast, an anonymous-variable is never bound to a value, i.e. it only acts as a 'place-holder' in the pattern. The wild card matches any continuous part of the score. Finally, an index-variable extracts values from an absolute position. Below we give some pattern-matching examples with their respective bindings (in order to clarify the examples the pattern below the input is formatted with the help of spaces so that it matches its input):

```
input:   ( 1  2  3  4  5)
pattern: (?1  ?  ?  ?2  ?)
match:    ?1 = 1,  ?2 = 4

input:   (1  2  3  4  5)
pattern: (i1 i2      i5)
match:   i1 = 1, i2 = 2, i5 = 5

input:   (1  2  3  4  5)
pattern: (*          ?1)
match:    * = (1  2  3  4), ?1 = 5

input:   (1  2  3  4)
pattern: (*    ?1 ?2)
match:    * = (1  2), ?1 = 3, ?2 = 4

input:   (1  2  3  4  5  6  7)
pattern: (?1 *          ?2 ?3)
```

```
match:    * = (2 3 4 5), ?1 = 1, ?2 = 6, ?3 = 7
```

For example an analysis rule that adds an expression denoting the interval between all adjacent melodic pitches found in a score can be written as follows (this rule uses a wild card and two variables):

```
(* ?1 ?2                 ;; pattern-matching part
  (?if                   ;; Lisp-code part
    (add-expression 'group ?1 :info (- (m ?2) (m ?1)))
  "interval rule")       ;; documentation string
```

In the old syntax the variables in the pattern-matching part always refer to note objects. This scheme has been useful as the note objects of the input score contain potential information of the current musical context, such as instrument, part, pitch, metrical position and harmony. The expressions '(m ?1)' and '(m ?2)' denote the pitch-values of the notes referred by the variables '?1' and '?2' ('m' stands here for 'midi').

While the old syntax has proven to be useful rules can occasionally be quite hard to define and understand. One reason for this difficulty is the fact that the variables always refer to the most low-level entity of the score (i.e. note objects), even if the rule works with more high-level concepts like chords, beats, measures or harmonies. Another difficulty is that the system strongly prefers melodic formations and the pattern-matching part is useful mainly when writing melodic rules. If the user wants to write for instance a harmonic rule then the required structural information has to be accessed in the Lisp-code part by using special Lisp help functions. Thus when writing non-melodic rules the user has to have knowledge about the implementation details of the system and master a large library of help functions. Finally, as one has to be conscious of implementation details the coding of new rules or Lisp help functions can become a complex and error prone process.

3. SCORE ACCESSOR SYNTAX

This section presents a new pattern-matching syntax that allows the user to specify special *score accessors*, which provide the user with a more high-level and intuitive approach when working with analytical rules. The variables given in the pattern-matching part of a rule can now refer to the structural entity the user is interested in, such as note, chord, beat, measure and harmony. The novel compiler is modular and new score accessors can be added incrementally.

The new syntax has many benefits: rules tend to be more compact and simple; the pattern matching language can be used systematically for all kinds of valid accessors; the user typically needs to know only a handful primitives in order to write new rules; potential bugs can be localized more easily as the most complex part of the system is localized in the compiler and not in user code.

In the new syntax the compiler accepts an optional score accessor keyword that defines the type of the variables that are declared in the pattern-matching part. If the rule

has no accessor keyword then the compiler assumes that the variables refer to note objects. For instance in the `"interval rule"` rule the variables '?1' and '?2' are notes. Valid accessor keywords are: ':chord', ':beat', ':measure', ':harmony', and ':score-sort'. This list can be extended by adding appropriate compiler methods for the new accessor keyword.

Another important change in our syntax is the function 'm'—which was already mentioned in the previous rule examples—that used to return the pitch-value (i.e. the MIDI key-number) of a note. The current version of 'm' is now more general as it can return besides pitch-values also lists and objects (thus 'm' stands henceforth for 'multi-accessor'). Now 'm' is defined as a method where the receiver or the first argument can be any variable type defined by the score accessor keyword. This means that the behaviour of 'm' depends on its first argument. For instance if the first argument is a note then 'm' returns a single numeric value; if the first argument is ':chord', ':beat', ':measure', or ':harmony' then 'm' returns a list of pitch-values. The 'm' method accepts also a number of optional keyword arguments that greatly enhances the functionality of this method. Keyword arguments can be used either to modify or filter the result or they can return a flag (i.e. either true or false). Currently the following keyword arguments are supported: ':int', ':harm-int', ':min', ':max', ':part', 'attack-item', ':complete-case?', ':object' and ':prev-item'. For instance the ':int' keyword allows to convert a list of pitch-values into a list of intervals; ':min' results in the minimum and ':max' in the maximum value of a pitch-value list; ':part' filters pitch-values so that 'm' returns only a pitch-value belonging to a given part.

The ':complete-case?' keyword relates to one specific concept in our system and needs some further explanation. When working with compound structures consisting of several notes such as chords, beats, measures and harmonic formations, the system calls the rules each time it encounters a new note in the structure. This scheme is due to the way the generative part of our system works. There a backtrack search is always working with *partial solution*s (for more explanation see [10]). Using the same scheme both in a generative and analytical context allows us to utilize rules interchangeably in both approaches without excessive rewriting. Thus the rules have to be able to deal with *partial formations* (i.e. cases where only some of the notes belonging to the current structure have been investigated). Normally this is not a problem as the 'm' method by default returns only values that have been considered. The 'complete-case?' keyword is typically used only in rules where it is required that the structure is fully present.

4. RULE EXPRESSIONS

ENP provides a collection of standard and non-standard notational attributes (e.g. articulations) called *ENP-expressions*. Furthermore, it offers a set of attributes that can be used to represent analytical information or other user-defined annotations as a part of a musical texture. In addition to their traditional use, ENP-expressions can be used in a wide range of applications: (1) display music theoretical analysis information, e.g, annotate motives, harmonic progressions; (2) visualize specialized analytical information, such as Schenker graphs, or pitch-class set theoretical information; (3) attach arbitrary textual annotations, names, or comments to objects; (4) dynamically inspect and visualize data contained in notational objects, such as pitch, intervals duration, velocity, and start-time.

All ENP-Expressions can access the data contained by the notational objects they are associated with (their musical context). This allows to design dynamic expressions that can automatically display relevant information about themselves and their musical context.

It is also possible to use a scripting language called ENP Script [7] as an algorithmic complement to the manual approach where the user inserts ENP-expressions by hand. This is useful when building, for example, computer-assisted music analysis applications.

In this article we focus on two expressions in particular, namely *score-expressions* and *analysis-groups*.

Score-expressions can be used to visualize discontinuous information in the score (Figure 1). They provide a convenient way to display information with some common identity that is scattered across different parts. Arbitrary vertical/horizontal relations can be made visible. The shape of a score-expression can be selected from a predefined collection. In addition, score-expressions can also contain user definable textual information. The following example enumerates the current set of different shapes (Figure 1):
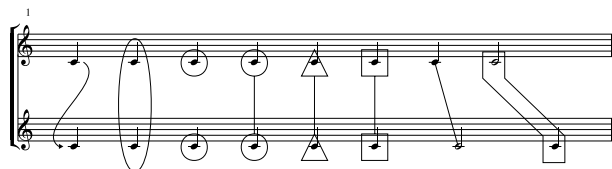

Figure 1. The available shapes of score-expressions displayed in an ENP score. The names of the shapes are: :curve, :oval, :encircled, :connected-shapes (with three different shapes :circle, :triangle, and :rectangle), :line and :path.

An analysis-expression, in turn, has the ability to keep a history and compare the similarity of its musical context to the ones of its previous occurrences. There are two important parameters associated with analysis-expressions: :name and :analysis-key. The analysis-expressions that share the same :name make observations about one musical quality at a time. This can be, for example, the interval between two consecutive notes, the pitch-class set-theoretical identity of a group of notes, etc. The :analysis-key, in turn, defines the algorithm that is applied to the musical context of the expression to calculate a special key which is then used by the system to compare the musical context with other occurrences of analysis-expressions that have the

same ₙₐₘₑ parameter. There is also an additional, third, parameter, :display-type, that can be used to define the graphical appearance of the expression.

Let us say that we have inserted several analysis-expressions in the score that are named :motive1. These expressions also contain an :analysis-key that is defined with the Lisp expression '(- (m ?2) (m ?1))'. All these expression share the same history (due to name). Each of these expression compare their musical context to the recorded history to determine if there already is a similar occurrence or not. The history is arranged so that similar musical contexts get similar tags (such as A, B, C, etc.). Thus expressions that have some similar qualities in their musical context also share similar graphical appearance.

Next we give an example of the behavior and properties of analysis-expressions. The corresponding ENP score is given below the analysis rule (Figure 2). Here we analyze the interval between two consecutive notes in the score and display the information in the score by a letter and we also draw a bracket that encloses the extent of the motive.

We can see that the expression has determined that out of all the consecutive intervals in this excerpt six are downward minor seconds (all marked with a letter B) and one of the intervals is a downward minor third (marked as A in the score).

```
(* ?1 ?2
  (?if
   (when (zerop (mod (1- (mindex ?2)) 2))
    (add-expression 'analysis-group ?1 ?2
                    :name :motives
                    :display-type :letter
                    :analysis-key (- (m ?2) (m ?1))))))
```


Figure 2. Motive analysis applied to a passage of music with the help of an analysis-group expression (J.S. Bach: Das Wohltemperierte Klavier I, Prelude and Fugue in b minor, BWV 869).

5. MELODIC RULES

In the following sections we will demonstrate how score accessors can be used to write analytical rules. Section 2 gave already one simple melodic rule example without score accessor keywords (i.e. in this case we assume that all variables in the pattern-matching part refer to notes).

As our first example we use the melodic analysis rule given in Section 2 to add textual information in a twelve-tone row (Figure 3). In this case we display the interval between any two consecutive notes directly in the score by using a group expression.

```
(* ?1 ?2
  (?if
   (add-expression 'group ?1 :info (- (m ?2) (m ?1))))
 "interval rule")
```


Figure 3. Interval information inserted in a twelve-tone row with the help of an analysis rule and group expressions.

Our next example (Figure 4) uses the accessor ':beat' and thus the variable named '?1' refers to a beat. This rule is applied to all beats in the input score. The 'm' method returns all pitch-values for all notes of the current beat. The function 'setp' checks that these pitches do not contain modulo 12 duplicates. The rule given below maps through all the beats in the score and inserts a red oval where the beat contains pitch-class duplicates.

```
(* ?1 :beat
  (?if
   (unless (setp (m ?1) :key 'mod12)
     (add-expression 'score-expression
                     (m ?1 :object :note)
                     :kind :oval
                     :color :red
                     :note-expression-p t)))

 "check for PC duplicates in beats")
```


Figure 4. The beats in the score checked with the help of an analysis rule. Beats with pitch-class duplicates are marked with a red oval.

6. HARMONIC RULES

The score accessor keyword ':harmony' allows to formulate both harmonic and voice-leading rules in a compact manner. In this section we focus in an example that deals with harmony. With the term harmony we mean here vertical pitch formations in the score where one, two or more notes are sounding together. The following two examples illustrate in a musical score the harmony concept in PWConstraints. The first example is a simple case of two homo-rhythmic parts with four harmonic formations (Figure 5). The notes belonging to each of the harmonies are encircled and connected with a line.
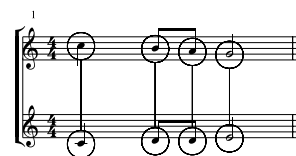

Figure 5. Four harmonies visualized in an ENP score with the help of connected circle shaped expressions.

Our second example is rhythmically more complex than the previous one resulting in more complex harmonic relations. Again, there are four harmonies but this time one note belongs to two different harmonies as can be seen in Figure 6. The harmonies are marked in the score

as above and each line can be considered to represent one harmony. The line connects the members of each of the harmonies and the members, in turn, are marked with circles.
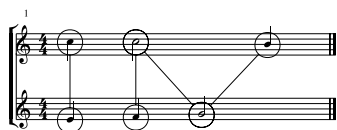

Figure 6. An example where one note is shared with two harmonies.

Let us assume that the assignment is to write three part choral voicing for a string trio using only complete triads, i.e., it is not allowed to duplicate the root, third or fifth of a triad. We create next a rule—by using the ':harmony' score accessor—that analyses a score according to a given harmonic aspect and marks the mistakes in the score in some meaningful way. In this case we use a score-expression that draws a path around the notes it is associated with. The rule is defined as follows:

```
(* ?1 :harmony
   (?if
    (let ((midis (m ?1 :complete-case? t)))
      (when (and midis (not (setp midis :key #'mod12)))
        (add-expression 'score-expression ?1
                    :kind :path
                    :info (format () "Incomplete~%triad!")
                    :color :red
                    :note-expression-p t)))))
```

Figure 7 shows one potential solution and also displays the information inserted in the score by the 'harmony teacher' rule given above. The mistakes are highlighted using a red rectangle along with the written instruction "Incomplete triad!" placed directly above it.
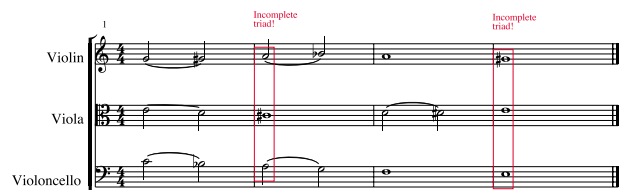

Figure 7. Incomplete chords marked in the score with the help of a 'harmony teacher' rule.

7. VOICE-LEADING RULES

Voice-leading rules tend to be harder to formulate than melodic or harmonic ones as they often deal both with melodic and harmonic formations in the same rule. The required musical context can be spread among several parts in the score. This can lead to difficult situations especially if the rhythmic structure of the input score contains complex poly-rhythms. In this section we continue to use the ':harmony' score accessor as it often allows to capture these cases in a compact manner.
Figure 8 gives a relatively complex excerpt taken from the piano repertoire (Sonata op. 58 by F. Chopin). The example contains clef changes which makes it more difficult to follow voice-leading. The rule given below

is used to check the piano texture and mark such places where a note in one part crosses or forms an unison with a note in another part. As can be seen in Figure 8 there is one such incident found in the score which is marked by encircling the offending note. This kind of knowledge can be beneficial when analysing, for example, piano texture. The analysis reveals that there is a conflict between the written and sounding representation of the piece. Albeit there is a quarter note sounding in the left hand part, in the right hand part there is an attack on the same note.

```
(* ?1 :harmony
   (?if
    (let* ((p1 (partnum ?csv)) (p2 (+ p1 1))
           (m1 (m ?1 :part p1)) (m2 (m ?1 :part p2)))
      (when (and m1 m2)
        (when (<= m1 m2)
          (add-expression 'score-expression ?csv
                      :kind :encircled
                      :note-expression-p t)))))
   "no voice crossings")
```
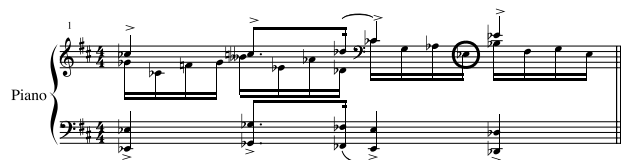

Figure 8. A rule that checks for voice crossings in a musical score. The encircled note in the second voice in the right hand part crosses with the bass part (F. Chopin: Sonata op. 58).

The following rules are used to check for modulo 12 cross-relations between the soprano and bass parts within two adjacent harmonic formations. If a cross-relation is found, a line-shaped score-expression is inserted in the score to visually reveal the associated notes (Figure 9).

```
(* ?1 ?2 :harmony
   (?if (when (m ?2 :complete-case? t)
          (let* ((sop1 (m ?1 :max t))
                 (bas2 (m ?2 :min t)))
            (unless (/= (mod12 sop1) (mod12 bas2))
              (add-expression 'score-expression
                  (m ?1 :max t :object :note)
                  (m ?2 :min t :object :note)
                  :note-expression-p t
                  :kind :line)))))
   "no sop/bass mod12 cross-relation1")

(* ?1 ?2 :harmony
   (?if (when (m ?2 :complete-case? t)
          (let* ((sop2 (m ?2 :max t))
                 (bas1 (m ?1 :min t)))
            (unless (/= (mod12 sop2) (mod12 bas1))
              (add-expression 'score-expression
                  (m ?2 :max t :object :note)
                  (m ?1 :min t :object :note)
                  :note-expression-p t
                  :kind :line)))))
   "no sop/bass mod12 cross-relation2")
```
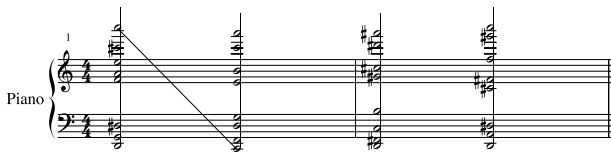
Figure 9. A cross-relation revealed in a sequence of chords.

8. ANALYSIS EXAMPLE

In the Appendix we give an analysis example taken from the literature. We show how to analyze the harmonic and melodic content of a choral piece, called 'Erotessa', by a Finnish composer Kimmo Kuitunen (one of the authors of this article). In our musical excerpt the cycles of three 4-voiced symmetric chords and their inversions (having the set-class identity 4-7, 4-7 and 4-9) create the harmonic basis. The pitch-classes remain the same within each four repetitions of the cycle, but the octave positions of pitch-classes are changed. These three chords (F-A-E-Ab , D-Bb-Eb-B and Db-Gb-G-C) also form a matrix which fills the chroma (12-1). The melodic line of each vocal part repeats the 3-note cells derived from the matrix (set-classes are either 3-3a or 3-3b). Each voice also uses all the 12 pitch-classes and thus fills the chroma. We use the graphical devices described above to visualize the analysis information directly in the score. We begin by giving the analysis rules applied in the example.

A.

```
(* ?1 :harmony
  (?if
   (let ((midis (m ?1 :complete-case? t)))
     (when (and midis (sym-chord? midis))
       (add-expression 'score-expression ?1
                       :kind :connected-shapes
                       :shape (case
                         (mod (length (all-prev-items ?1)) 3)
                           (1 :rectangle)
                           (0 :circle)
                           (2 :triangle))
                       :note-expression-p t)))))
```

B.

```
(* ?1 ?2 ?3
  (?if
   (when (zerop (mod (1- (mindex ?1)) 3))
     (add-expression 'analysis-group ?1 ?2 ?3
                     :name :motives
                     :display-type :letter
                     :analysis-key
                     (list (mod12 (m ?1))
                           (mod12 (m ?2))
                           (mod12 (m ?3)))))))
```

Let us examine the analysis rules in more detail. The first one of the analysis rules (A) deals with harmonic formations of the example score. We use the rule to locate any symmetrical vertical formations in the score. In order to be able to make this kind of analytical observations about the harmony, we must first ensure

that the current harmony is fully present (see the expression '(midis (m ?1 :complete-case? t))' in the rule). If the harmony is not complete, the midis variable is an empty list in which case the latter part of the rule is not executed. However, if midis contains some values these are checked by the function sym-chord? to determine if they constitute a symmetric chord. If this is true then a score-expression (:connected-shapes) is inserted in the score to visually reveal the notes that belong to the symmetric chord. In order to distinguish between overlapping situations the shape of the score-expression is rotated between three different shapes: :rectangle, :circle, and :triangle. This is done according to the position of the current harmony in the score (see '(mod (length (all-prev-items ?1)) 3)' in the rule). Note that the shapes do not have any meaning of sameness in an analytical sense. The shapes are used only to help distinguishing between several overlapping expressions. One option, however, could have been to use the visual appearance to identify, for example, some pitch-class set-theoretical properties by applying similar shapes to similar harmonies.

In the second analysis rule (B) we perform automatic motive analysis of the score. This is done by using the ENP-expression called analysis-group that was discussed in Section 4. In this case we are interested in the motive identity of 3-note groups. As can bee seen in the score, this particular example is quite clearly organized from this point of view. We get only four different motives (marked as A, B, C, and D). The motives also appear to have a cyclic pattern which can be observed by examining each part individually.

9. CONCLUSIONS

This paper presents a new syntax that allows to write in a flexible manner analytical rules that are applicable to a wide range of musical contexts. The pattern-matching part can be used systematically for all score accessor types. Analysis rules return expression objects that are used to visualize analytical data.

Although the system is still under development it is already obvious that this kind of approach has many interesting applications in computer-assisted composition, analysis and teaching contexts.

10. ACKNOWLEDGEMENTS

11. REFERENCES

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

[2] Rueda C., M. Lindberg, M. Laurson, G. Bloch, and G. Assayag. "Integrating Constraint Programming in Visual Musical Composition Languages", in *ECAI 98 Workshop on*

Constraints for Artistic Applications, Brighton, 1998.

[3] Anders T. "Arno: Constraints Programming in Common Music", *Proceedings of the International Computer Music Conference*, 2000.

[4] Truchet C., G. Assayag, and P. Codognet. "Visual and Adaptive Constraint Programming in Music", *Proceedings of the International Computer Music Conference*, Havana, Cuba, pp. 346-352 , 2001.

[5] Anders, T. *Composing Music by Composing Rules: Computer Aided Composition employing Constraint Logic Programming.* Sonic Arts Research Centre Queens University Belfast, Northern Ireland, 2003.

[6] Laurson, M., and M. Kuuskankare. "PWGL: A Novel Visual Language based on Common Lisp, CLOS and OpenGL", *Proceedings of the International Computer Music Conference.* Gothenburg, Sweden, pp. 142–145, 2002.

[7] Kuuskankare, M. and M. Laurson. "Intelligent Scripting in ENP using PWConstraints", *Proceedings of the International Computer Music Conference.* Miami, USA, pp. 684-687, 2004.

[8] Huron D. "Music information processing using the Humdrum Toolkit: Concepts, examples, and lessons", *Computer Music Journal.* Vol. 26, No. 2, pp.15-30, 2002.

[9] 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, pp. 463-466, 2002.

[10] Laurson, M., and M. Kuuskankare. "Extensible Constraint Syntax Through Score Accessors", In *Journées d'Informatique Musicale 2005.* Paris, France, 2005.

APPENDIX

Kimmo Kuitunen: Erotessa (lyrics by Eino Leino). The tenor part sounds as written. Symmetric harmonies are displayed in the score with the help of score-expressions. Some motive analysis information is also shown below each part (marked with letters A to D).