Revista Vórtex | Vortex Music Journal | ISSN 2317–9937 | http://vortex.unespar.edu.br/
D.O.I.: https://doi.org/10.33871/23179937.2022.10.1.4

Real-time Analysis of Interactive Scores in PWGL

Mika Kuuskankare

Independent grant researcher | Finland

Abstract: In this article, we introduce an original approach to computerized music analysis within the graphical computer-assisted composition environment called PWGL. Our aim is to facilitate real-time analysis of interactive scores written in common Western music notation. To this end, we have developed a novel library that allows us to analyze scores realized with the help of ENP, and to visualize the results of the analysis in real-time. ENP is the native music notation tool of PWGL able to produce automatically typeset and interactive music notation. Here, it is extended to support the display of analytical information that can be drawn on top of the score as an overlay. The analysis backend is realized with the help of our built-in musical scripting language. The language is based on pattern-matching and allows for a rich access of score information. The results of the analysis are presented directly as a part of the original score leveraging the extensible and interactive visualization capabilities of ENP.

Keywords: computational musicology, soft real-time systems, computer-assisted composition, interactive music notation, music analysis.

Received on: September 29th, 2021. Approved on: October 27th, 2021. Online available on: April 30th, 2022. Presented at the 18th Brazilian Symposium on Computer Music (2021). Guest editors: Dr. Ivan Eiji Simurra (UFAC), Dr. Flávio Schiavoni (UFSJ), Dr. Cristiano Figueiró (UFBA), Dra. Isabel Nogueira (UFRGS) e Dr. Tiago Tavares (Unicamp). Creative Commons CC-BY-NC.

In computational musicology, it is customary to use non-real-time algorithms to analyze music encoded in textual form. Within the computer music community, real-time music analysis has become somewhat synonymous to real-time audio analysis and visualization. This article, by contrast, focuses on real-time analysis of interactive scores written in common Western music notation. This approach is still underrepresented within the community mainly because of the computational complexity associated with these operations.

It is customary to divide real-time systems into two main categories: *hard real-time* and *soft real-time*. Systems falling under the first category usually present either safety or mission critical constraints and, in general, missing a deadline is considered disastrous. Systems in the second category work "fast enough" and missed deadlines generally affect only the quality of service. This article won't debate the fine points of real-time computing. Instead, we consider the latter category, specifically in the sense of fast–or fast enough–computation.

To frame the present problem, let us consider the representations of music in state-of-the-art real-time and non-real-time musical environments. Real-time environments, such as PureData (PUCKETTE, 2002), concentrate on producing and manipulating sound, video, and graphics, rarely providing robust interactive representations of musical notation. Non-real-time environments, such as OpenMusic (BRESSON; AGON, 2011) for computer-assisted composition or LilyPond (NIENHUYS; NIEUWENHUIZEN, 2003) for music notation, allow for robust musical representations at the expense of interactivity. However, rich music representations and real-time interactivity need not to be mutually exclusive, instead, the two should complement each other. Hampering effects are easily noticed when analyzing at how other fields have solved the issue. Text editors, for example, allow people to develop documents in real-time while retaining a rich view of these documents. The same is true of photo editing, where users can apply sophisticated filters to their photos and instantly see the results.

Music notation has evolved into a rich visual and semantic system. Encapsulating the notational and analytical practices of the western art music in a real-time context, requires versatile and interactive representations of music, flexible visualization devices, and musical knowledge encoded into the system. The majority of music notation systems are unable to cope with the requirements of music analysis. The designs and internal representations of the systems do not adequately encompass the scope and variety of music analytical notation (HAMEL, 1989). The systems almost invariably prevent access to the internal structures. They are also unable to present the analytical information in an interactive way, i.e., updating in real-time according to the changes in the musical content.

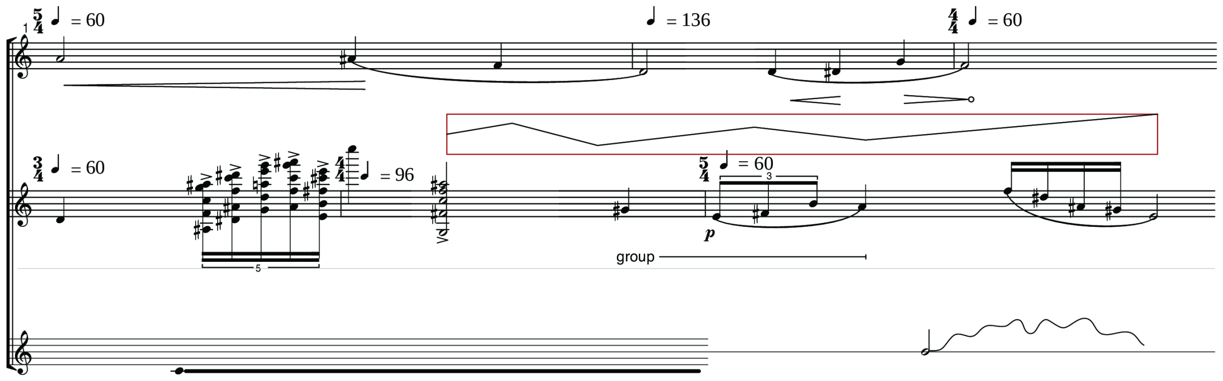
PWGL (LAURSON; KUUSKANKARE; NORILO, 2009) is a visual music programming language written in Lisp and OpenGL. Its primary focus is on computer-assisted composition in an integrated environment with music notation, analysis, performance and software synthesis. Currently, it can be used to analyze musical scores realized with the help ENP (KUUSKANKARE; LAURSON, 2006) using a variety of methods as reported in LAURSON, KUUSKANKARE AND KUITUNEN (2005, 2008), KUUSKANKARE (2013) and KUUSKANKARE AND SAPP (2013) among others. However, these methods have not yet been adapted in a real-time context. In this article, we present a novel PWGL library, that aims to make it possible to analyze ENP scores and to visualize the results of the analysis in real-time. The analysis backend is realized with the help of ENP-Script (KUUSKANKARE; LAURSON, 2004). ENP is PWGL's built-in music notation program developed primarily for applications concerning computer-assisted composition and virtual instrument control. ENP-Script, in turn, is the native scripting language of ENP. It allows us to access complex musical patterns with the help of a pattern-matching syntax developed for our constraint-based compositional system.

The rest of the article is organized as follows. First, we introduce the two key software components of the library: ENP and ENP-Script. Next, we discuss implementation details and present some examples of real-time analysis rules. Finally, we present preliminary data on the performance of our system and end with some concluding remarks and ideas for future work.

1. ENP

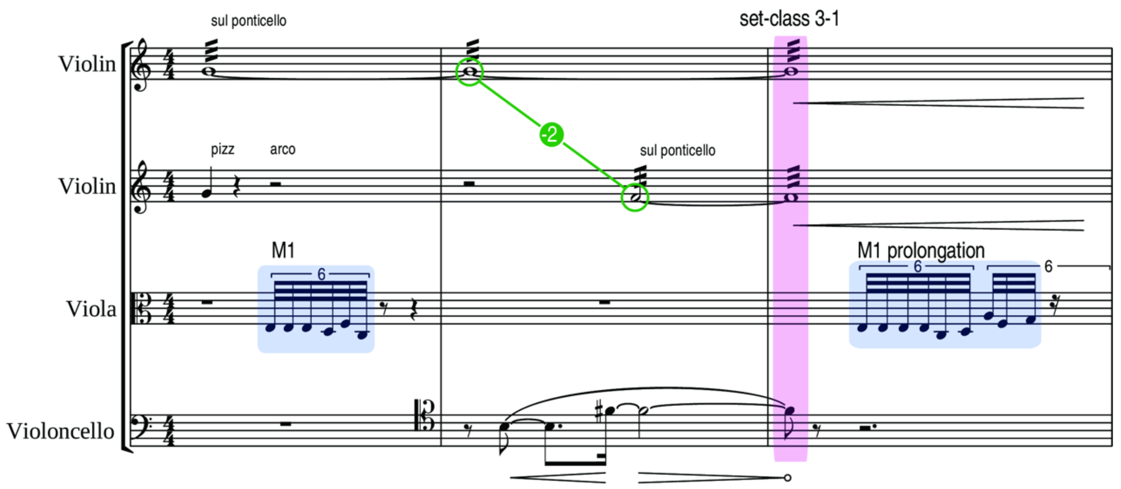
ENP is a music notation tool within PWGL. It can produce automatically typeset and interactive music notation as can be seen in Figure 1. ENP has a graphical user-interface supporting direct editing. It implements an extensive set of graphical devices (called ENP-Expressions) for enriching notational output. ENP provides full access to the musical structures behind the notation and allows for algorithmic control. It also provides the users with the possibility to extend and customize the notational output. This can be done either algorithmically or using any of the built-in visual tools.

FIGURE 1 – A relatively complex score notated with the help of ENP. Besides standard notational devices, such as articulations and dynamics, the score showcases several types of graphical control information as well as different simultaneous tempi and time signatures.



At the center of the expressive power of ENP are interactive and user-definable graphical devices, called ENP-Expressions. In ENP, the term expression is used in a broad sense: expressions are fully programmable visualization devices that can be used to represent dynamic Lisp-based objects as a part of a musical texture (see Figure 2). In addition to their traditional use in representing articulations and other standard markings, they can be used in a wide range of applications, such as providing visual control information for virtual instrument synthesis (LAURSON; NORILO; KUUSKANKARE, 2005) or representing analytical information. One of the unique features of ENP-Expressions is that they can be attached to a discontinuous group of objects spanning across multiple parts. This is especially beneficial in musical analysis applications that typically deal with the relationships between independent melodic lines and simultaneously sounding harmonies.

FIGURE 2 – An example of different kinds of ENP-expressions. Alongside the traditional expression markings, such as slurs and dynamics, we use here a combination of text and colored areas to highlight motives as well as commonly accepted music theoretical markings, such as set class names to display the total harmonic content of select simultaneities.

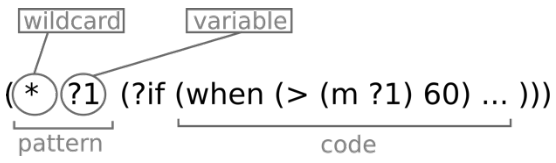


2. ENP-Script

ENP-Script is the scripting language of ENP. It uses the Lisp-based syntax of our pattern-matching language, PWConstraints (LAURSON; 1993). PWConstraints is a compositional system that uses the Backtracking algorithm to obtain a solution to musical constraint satisfaction problems. ENP-Script, while sharing most of its behavior with PWConstraints (e.g., language syntax, score access), does not use backtracking.

The scripts consist of one or more scripting rules. The scripting rule, in turn, consists of a pattern-matching part written in our custom pattern-matching language and a code part written in Lisp (see Figure 3). The pattern-matching part of the rule is defined using a set of wildcards and pattern-matching variables. Each variable represents an object in the score and holds information about the associated notes including pitch, rhythm, rhythmic position, current harmony, etc. Once the pattern-matching part has accessed the desired score objects (e.g., notes, chords, measures, harmonic formations), these variables can be used in the code part to access score information. Here, the 'm' method is typically used ('m' stands for multi-accessor). Normally, this method returns either a single note or a list of notes for compound objects (e.g., chords, beats, measures, harmonic formations). It also accepts a list of keyword arguments. These arguments allow us to specify more precisely the type and structure of data that is returned. Finally, the scripting engine loops through the score objects (usually notes) or some pre-generated compound object structures (e.g., harmonies or voice-leading data) and applies the scripting rules to produce some effect, such as attaching an expression. See KUUSKANKARE AND LAURSON (2004) for a more detailed discussion of ENP-Script.

FIGURE 3 – A typical scripting rule consists of the pattern-matching part (to the left) and the code part (to the right). The pattern-matching part and the code part are separated by the mandatory '?if' token.



3. The Real-time Analysis Library

The library contains two modules: (1) the visualization module, and (2) the analysis module. The analysis results are visualized directly on the source score. The analyze-visualize cycle is triggered every time a property of some score object is changed. This typically happens when the user, for example, transposes a note or edits the rhythm.

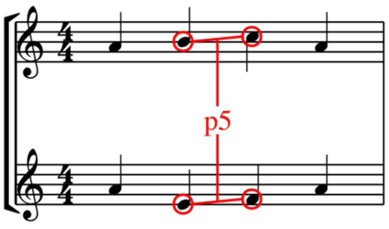
3.1. The Visualization Module

For the purposes of this library, a new category of ENP-Expressions–called overlay-expression–was developed. These expressions differ from standard ones in a few specific ways: (1) they are not part of the object structure of the score, i.e., they are not attached to any score object or saved with the associated score, and (2) they are temporary, i.e., the lifetime of the overlay objects ends before the next batch of objects begins to be initialized.

As the name suggests, overlay-expressions are drawn as an overlay on top of existing score objects. Most expressions, such as dynamics or articulations, are usually attached to objects in one single part, therefore relying on the local coordinate system of that part. However, an overlay-expression refers to the global coordinates (i.e. anywhere on the current page) of the objects it is associated with and, therefore, can easily extend itself across different parts, as can be seen in Figure 4.

Currently, the library implements two expressions that extend the overlay-expression: the *voice-leading expression* and the *voicing expression*. Figure 4 shows the voice-leading expression. It holds a reference to a pair of notes in two different parts. Visually, it connects the notes in each part with a line that has a hollow circle on each end. Furthermore, a vertical line connects the note pairs involved in the offending voice leading case and displays an error code around the midpoint of the line segment. The error code is user-definable.

FIGURE 4 – The voice-leading expression indicating the notes involved as well as the name of the error. In this case, the error code 'P5' denotes a parallel fifth.



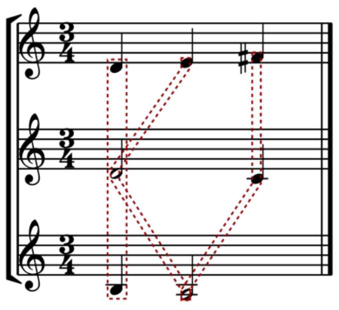
The voicing-expression marks cases in the score where there is either: (1) a voice crossing, or (2) an open spacing. The voicing-expression holds a reference to the corresponding notes that, by definition, have to be in two different parts. The two notes are encircled and connected vertically with a dotted line. The user-definable error code is again indicated around the midpoint of the line segment (see Figure 7 for an example).

The expression system of ENP is fully user-programmable. Therefore, the users can extend the repertoire of these real-time analytical expressions or adapt the existing ones to suit their specific needs. This can be done programmatically by sub-classing the overlay-expression, or visually using our built-in expression designer tool (KUUSKANKARE; LAURSON, 2005).

3.2. The Analysis Module

The analysis module is responsible for performing the analysis and passing the analysis information to the visualization module. The analysis module consists of two parts: (1) the real-time analysis engine, and (2) the user-definable analysis rules. To facilitate real-time response, the real-time analysis engine implements a slightly modified version of our scripting engine. One of the differences is that the additional compound structures prepared in advance to assist in accessing the multitude of different score objects are cached instead of being regenerated every time a script is run. One of the most useful of these structures is the harmonic structure shown in Figure 5. It plays an important part in solving both contrapuntal and harmonic problems. Also, as a further optimization, the scripts are only applied to the part of the score that is visible to the user. This is typically the current page.

FIGURE 5 – To assist and optimize the search process, several auxiliary structures are generated for both our constraints and scripting engines. The simultaneity (i.e, harmonic) structure is shown here. The polygons enclose the three possible simultaneities in this particular score.



4. Some Rule Examples

In this section, we will examine in detail how to implement some basic rules for our real-time analysis engine. Specifically, we consider the detection of parallel fifths progressions, and symmetric harmonies.

4.1. Detecting Parallel Fifths

Consecutive parallel fifths are progressions in which the interval of a perfect fifth is followed by another perfect fifth between the same two parts. The algorithmic detection of this progression is relatively straightforward. The code is shown in Listing 1. In Line 1, we define the pattern and type of score objects (indicated by the keyword :harmony) to which this rule is applied. The pattern consists of a wildcard ('*') which matches zero or more score objects and a variable ('?1'). This particular pattern extracts all the harmonic formations (groups of simultaneously sounding notes) from the score and binds them one by one to the variable named '?1'. In line 3, we call the m-method with the keyword ':vl-matrix' as an argument to return an object that encapsulates the relevant notes that participate in all the possible voice leading cases between two consecutive simultaneities. In four part texture, this results in at most six different cases: voice 1 against voices 2, 3, and 4; voice 2 against voices 3 and 4; and voice 3 against voice 4. The presence of a rest in any of the voices naturally reduces the number of combinations. In line 5, we retrieve a list of notes that belong to the top-most voice. In the next line, we check that this voice actually moves up or down, otherwise it will not contribute to any parallel movement. Next, we access the pitch properties of the two notes and check if they are equal (line 6). In case the pitches are not equal, we need to loop through all the remaining voices (line 7). For each voice, we again check that it is moving in either direction (line 9), and, if so, proceed with the remaining checks. Lines 10-11 check that intervals between the two voices are both perfect fifths, while lines 12-13, in turn, check that both voices also move in the same direction.

If all the aforementioned conditions are satisfied, the two voices move in parallel fifths. To indicate this in the score, we instantiate an overlay-expression that is subsequently attached to the formation of the four notes (two notes in each of the involved parts). This is done in lines 14-17. The overlay-expressions have many user-definable properties, one of which is color. Here, we display the overlay in red (see line 17) to denote a high level of urgency.

LISTING 1 – Code for detecting parallel fifth progressions in an ENP score.

```
 1 (* ?1 :harmony
 2  (?if
 3   (let ((mat (matrix-access (m ?1 :vl-matrix 2 :object t) :h)))
 4     (when mat
 5       (destructuring-bind (upper-1 upper-2) (first mat)
 6         (unless (= (m upper-1) (m upper-2))
 7           (loop for mel2 in (rest mat) do
 8                 (destructuring-bind (lower-1 lower-2) mel2
 9                   (unless (= (m lower-1) (m lower-2))
10                     (when (and (= (mod12 (abs (- (m upper-1) (m lower-1)))) 7)
11                                (= (mod12 (abs (- (m upper-2) (m lower-2)))) 7)
12                                (= (signum (- (m upper-2) (m upper-1)))
13                                   (signum (- (m lower-2) (m lower-1)))))
14                       (add-overlay upper-1 upper-2 lower-1 lower-2
15                                    :kind :voice-leading
16                                    :id :parallel-fifth
17                                    :color :red))))))))))))
```

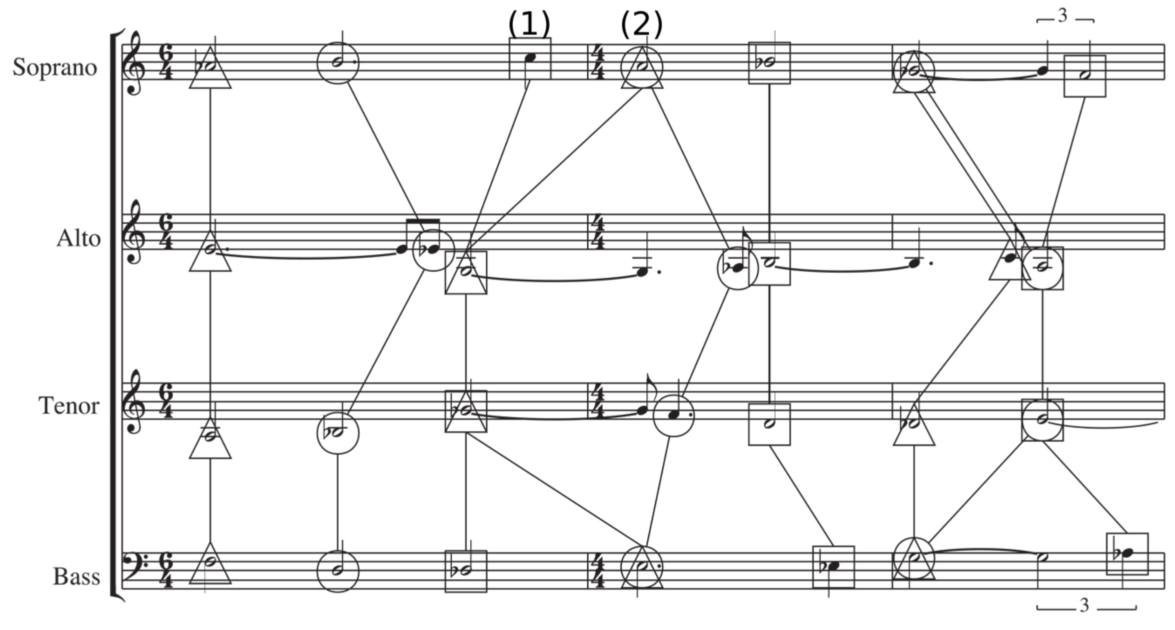
4.2. Detecting Symmetric Harmonies

Our next example allows us to detect and visualize symmetric harmonies in real-time (see Listing 2). Line 1 defines a pattern that maps all harmonies in the score. Again, the pattern consists of a wildcard ('*') and a variable ('?1'). In line 3, we access a list of MIDI note numbers from the given harmony and store them in the variable named 'midis'. In line 4, in turn, we setup a variable named ':shape'. The ':shape' variable behaves much like a static local variable. A static local variable is initialized only once and its value is retained and accessible in the function in which it is declared, e.g., the analysis rule in this case. Here, the value of ':shape' is used as a counter and incremented each time the rule is executed. Next, in line 7, we check if the MIDI note numbers stored in the 'midis' variable make a symmetric harmony using the built-in function called 'sym-chord?'. If so, we insert an overlay expression of type :connected-shapes (lines 8–14) to the score object represented by the variable named '?1'. Note, that this variable represents a complete harmony, which is a compound object holding a reference to an arbitrary number of note objects. To be able to clearly distinguish between consecutive symmetric harmonies, we cycle through 3 different shapes, namely circles, rectangles, and triangles (see lines 12–14). Figure 6 shows a score--by the Finnish composer Kimmo Kuitunen--that has been analyzed using the rule described above. As can be seen, symmetrical harmonies have been extensively used by the composer.

LISTING 2 – Code for detecting symmetric harmonies in an ENP score.

```
 1 (* ?1 :harmony
 2    (?if
 3     (let ((midis (m ?1 :complete-case? t))
 4           (shape (pmc-value
 5                  :shape
 6                  #.(let ((x -1)) #'(lambda () (incf x))))))
 7       (when (and midis (sym-chord? (m ?1)))
 8         (add-overlay ?1
 9                     :kind :connected-shapes
10                     :id :symmetric-harmony
11                     :shape (case (mod shape 3)
12                              (0 :circle)
13                              (1 :rectangle)
14                              (2 :triangle)))))))
```

FIGURE 6 – Symmetric harmonic structures marked in the score (Kimmo Kuitunen, "Erotessa" for mixed choir). Naturally, a note can belong to any number of consecutive symmetric harmonies, and, therefore, shapes are here used to distinguish between harmonies that potentially share notes. We use three different shapes: circles, rectangles, and triangles. Thus, for example, the third harmony (1) shares two notes with the next one (2). This is confirmed by the presence of the rectangle/triangle shape in the notes in the Alto and Tenor parts.

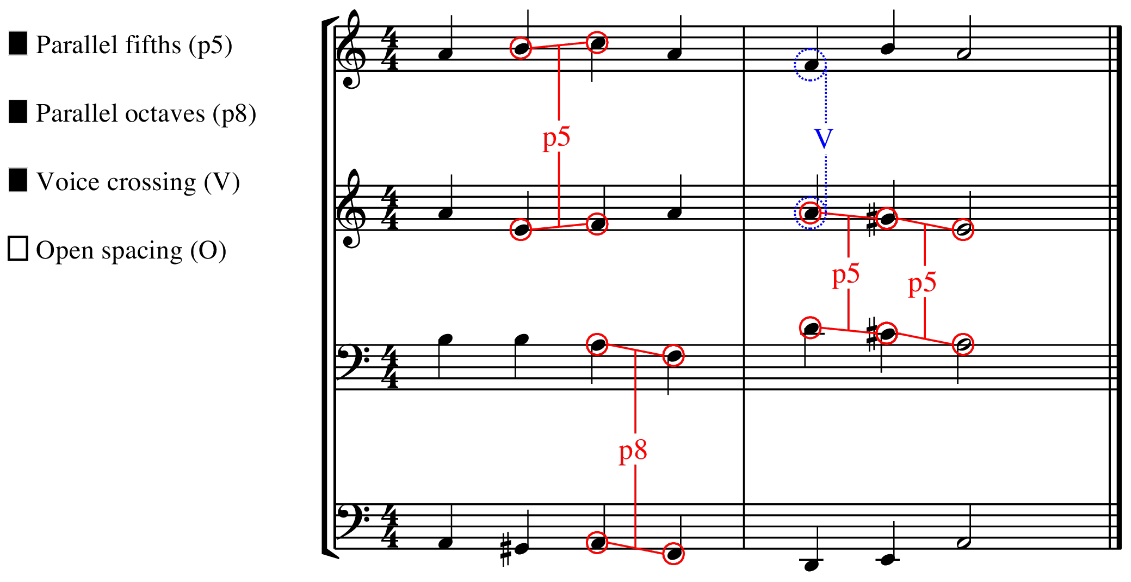


5. Performance Measures

In a real-time system, the time required for completion of the tasks is obviously important. We collected some performance measures with the help of an application prototype built specifically for testing purposes (see Figure 7). The test score is a four-part choral. Our test application implements the following rules: (1) parallel fifths and octaves, (2) voice crossing, and (3) open spacing. Parallel fifths and octaves are progressions in which the interval of a perfect fifth or an octave is followed by the same interval between the same two parts. The rule we are using here is a modified version of the one shown in Listing 1. Voice crossing is the intersection of melodic lines, e.g., a lower voice crosses above a higher one. Spacing, refers to the simultaneous vertical placement of notes in relation to each other. Open spacing typically occurs when the interval between consecutive voices exceeds an octave. Note, that the details about the interpretation of different rules typically varies from period to period. Currently, the rules are interpreted in the strictest sense without any exceptions or consideration to periodic conventions.

In the application, the names of the rules are displayed on the right accompanied with checkboxes (see Figure 7). The checkboxes can be used to activate or de-activate rules. Only errors selected by the user are checked and visualized and updated in real-time as the user works on the score. In addition to pitch, the user can manipulate any other score property, including rhythm.

FIGURE 7 – The test application displaying common voice-leading errors. The codes 'P5' and 'P8' indicate parallel fifths and octaves respectively and 'V', in turn, indicates the presence of voice crossing (the alto part crossing above the soprano part in this case).



Processor speed is obviously a factor in testing. To keep things in check, we chose a relatively old computer system for our test. The test application was run on a 2012 Apple MacBook Pro with an Intel Core i7 processor running at 2.7 GHz. The results are presented in Table 1. The 'total' time includes the initial layout calculation and drawing of the score (≈ 10 ms), the analysis phase (≈ 15 ms) and the drawing of the overlay objects (≈ 2 ms). The response time of around 30 ms seems to be in the realm of real-time and the GUI response confirms this. Thus, our experiment aligns well with our initial objectives.

TABLE 1 – The approximate timings of the steps involved in the real-time analysis process.

Step	**Time (ms)**
layout/draw	10
analyze	15[1]
visualize	2
total	27

[1] 35 ms without caching

6. Conclusions

In this article, we presented a novel PWGL library that allows for the real-time analysis of musical scores written in the ENP score format. The potential application fields range from elementary music pedagogy to advanced counterpoint exercises, as well as computer-assisted composition.

Our methods could be used, for example, in computer-based training, making it possible to develop interactive music theory applications. Being able to see the analysis in real-time should be an effective way to gain deeper understanding of how the elements of the composition (e.g., harmony, melody) work together. Furthermore, our tool could be used not only as a music analysis tool, but also as a compositional tool. Visual hints could assist the compositional process (e.g., keeping track of Harp pedaling) thus allowing the composer to focus on other tasks. Finally, our approach is open and extendable, and, therefore, the users can invent their own rules and visualizations to fit their particular needs.

There will undoubtably be ample opportunities for developing the functionality and performance of the library. The performance optimizations include, for example, precompiling and caching the analysis scripts. Currently, the rules are compiled every time a script is run. Furthermore, a visual interface for defining the scripts would be an interesting avenue for research. Finally, we are also working on additional analysis backends, one of which will be based on the Humdrum Toolkit (HURON, 2002). The Humdrum Toolkit is a widely-used open-source software package for musicological research developed at CCARH at Stanford University. It would allow us to access the rich and readily available music analysis features of Humdrum and a vast number of freely available pieces encoded in the kern data format. The different backends would not be mutually exclusive; instead, they would complement each other.

ACKNOWLEDGMENT

This work of Mika Kuuskankare has been supported by the Jenny and Antti Wihuri Foundation.

REFERENCES

BRESSON Jean; AGON Carlos. Visual programming and music score generation with OpenMusic. In IEEE Symposium on Visual Languages and Human-Centric Computing, 2011.

HAMEL Keith A. A design for music editing and printing software based on notational syntax. *Perspectives of New Music*, Vol. 27, No. 1, 70–83, 1989.

HURON David. Music information processing using the Humdrum Toolkit: Concepts, examples, and lessons. *Computer Music Journal*, Vol. 26, No. 2, 15–30, 2002.

KUUSKANKARE Mika. Schenkerian analysis tools in ENP. In Proceedings International Computer Music Conference, 2013.

KUUSKANKARE Mika; LAURSON Mikael. Intelligent Scripting in ENP using PWConstraints. In Proceedings of International Computer Music Conference, pages 684–687, 2004.

KUUSKANKARE Mika; LAURSON Mikael. ENP Expression Designer – a Visual Tool for Creating User Definable Expressions. In International Computer Music Conference, pages 307–310, 2005.

KUUSKANKARE Mika; LAURSON Mikael. Expressive Notation Package. *Computer Music Journal*, Vol. 30, No. 4, 67–79, 2006.

KUUSKANKARE Mika; SAPP Craig. Visual Humdrum-library for PWGL. In Proceedings of ISMIR, 2013.

LAURSON Mikael. PWConstraints. In G. Haus and I. Pighi, editors, X Colloquio di Informatica Musicale, pages 332–335, 1993.

LAURSON Mikael; KUUSKANKARE Mika; KUITUNEN Kimmo. Introduction to computer-assisted music analysis in PWGL. In Sound and Music Computing '05, 2005.

LAURSON Mikael; NORILO Vesa; KUUSKANKARE Mika. PWGLSynth: A Visual Synthesis Language for Virtual Instrument Design and Control. *Computer Music Journal*, Vol. 29, No. 3, 29–41, 2005.

LAURSON Mikael; KUUSKANKARE Mika; KUITUNEN Kimmo. The Visualization of Computer-assisted Music Analysis Information in PWGL. *Journal of New Music Research*, Vol. 37, No. 1, 61–76, 2008.

LAURSON Mikael; KUUSKANKARE Mika; NORILO Vesa. An Overview of PWGL, a Visual Programming Environment for Music. *Computer Music Journal*, Vol. 33, No. 1, 19–31, 2009.

NIENHUYS Han-Wen; NIEUWENHUIZEN Jan. LilyPond, a system for automated music engraving. In XIV Colloquium on Musical Informatics (XIV CIM 2003), 2003.

PUCKETTE Miller. Using Pd as a score language. In Proceedings of International Computer Music Conference, pages 184–187, 2002.

ABOUT THE AUTHOR

Dr. Mika Kuuskankare is a Finnish composer, with extensive background in music composition, research, and software engineering. Kuuskankare studied composition at the Sibelius Academy and received his Master's degree in Composition in 1999 and his doctorate in 2006. His compositional works include music for solo instruments, chamber ensembles and symphony and wind orchestras. His works are published by the Uusinta Publishing Company. During the past three decades, he has been involved in several interdisciplinary research projects that have resulted in important contributions to various areas computer music, and close to a hundred peer-reviewed publications. He was invited to do his research at IRCAM (Chercheur Invité, 2011) and at CCRMA, Stanford University (Visiting Scholar, 2012–2013). Kuuskankare is one of the originators of PWGL which is a graphical environment for computer-assisted composition. ORCID: https://orcid.org/0000-0001-8963-8104. E-mail: mkuuskan@yahoo.com