VISUAL HUMDRUM-LIBRARY FOR PWGL

Mika Kuuskankare
DocMus
Sibelius Academy
mkuuskan@siba.fi

Craig Stuart Sapp
CCARH
Stanford University
craig@ccrma.stanford.edu

ABSTRACT

We introduce a PWGL Humdrum interface that integrates command-line unix tools for music analysis into a visual programming environment. This symbiosis allows users access to the strengths of each system—algorithmic composition and visual programming components of PWGL along with computational analysis and data processing features of Humdrum tools. Our novel interface for Humdrum graphical programming allows non-programmers better access to Humdrum analysis tools, particularly with the built-in music notation display capabilities of PWGL. ENP (Expressive Notation Package) data from PWGL can be exported as Humdrum data. Humdrum files in turn can be converted back into ENP data, allowing bi-directional communication between the two software systems.

1. INTRODUCTION

PWGL [10] is a visual music programming language written in Common Lisp, CLOS (Common Lisp Object System) and OpenGL. Its primary focus is on computer-assisted composition in an integrated environment with music notation, software synthesis and music theory and analysis tools. PWGL comprises several large-scale applications, such as ENP [5] for music notation, PWGLConstraints [7] for rule-based composition, and Kronos [11] for sound synthesis.

The Humdrum Toolkit is a widely used open-source software package for musicological research conceived of by David Huron [1] in the 1990's and has been developed and extended by others. It is composed of two main components: the Humdrum data format and the Humdrum Toolkit that processes music in this format. The Humdrum file format is a generalized data array. Each column represents a stream of data (such as a part in a musical score), while each row represents simultaneous events across multiple data streams. A particular advantage of this format is that it allows inclusion of analytic data streams alongside the original musical score within the same file. The primary music-content subformat in Humdrum is called *kern*, with about 50 other subformats predefined in the standard toolkit for encoding musical features such as lyrics, dynamics, tuning, harmonic analysis and perceptual data. the Humdrum file structure also allows users to define their own subformats for specific analytic or markup purposes.

The Humdrum Toolkit is a collection of command-line utilities operating on data written in the Humdrum syntax. In addition to the standard Humdrum utilities that are written in AWK, a software package called *Humdrum Extras* developed at CCARH at Stanford University extends and compliments the original toolset with additional command-line tools and a C++ code library for manipulating data in the Humdrum syntax with a focus on efficient numeric processing of the data. [1] Humdrum data processing can also done within PERL [2], and kern data can be imported into the music21 system implemented within Python. [2]

PWGL can be used to analyze scores written in the ENP format by using a built-in rule-based scripting language called ENP-script [4]. ENP offers both an extensive and extensible set of visualization devices that can be used to visualize analytic observations directly in the score [6]. It has already been used for music analysis, as reported for example in [9] and [8]; however, the current approach is not well suited for statistical analyses. Furthermore, a scant amount of musical data is readily available in the ENP format. The PWGL Humdrum interface now enables access to a vast number of pieces encoded in the kern data format that is made freely available for download from `http://kern.ccarh.org` [12]. In addition to direct translation of kern data into ENP, the interface allows access to a large number of predefined analytical tools, and allows for the batch analysis of a large musical corpus at once.

Our library attempts to make it easier to work within the Humdrum environment. First, it makes it possible for non-programmers to use and access Humdrum analysis tools through a visual interface. Many theorists, musicians, and composers may be intimidated by Humdrum's unix-flavored command-line syntax thus limiting the number of potential users. Second, for musicians it is important to be able to see the piece of music they are working on in common music notation. ENP allows an instant, editable and extendable representation of thousands of pieces of music encoded in kern notation. Finally, the analytic in-

[1] http://extras.humdrum.org and http://github.com/craigsapp/humextra
[2] http://mit.edu/music21

formation obtained via Humdrum tools can be visualized directly within an ENP score which assists interpretation of the results.

Previous attempts to create visual interfaces around Humdrum tools, such as JRing [3] have been made; however, the approach provided by PWGL is more general and user-extendable since PWGL is a generalized programming environment. The combination of Humdrum and PWGL should prove especially beneficial for musicologists, but should be of interest to MIR researchers and the like as well.

2. HUMDRUM LIBRARY

The PWGL Humdrum library is implemented in two parts: kern import is implemented as a Humdrum command-line tool called *hum2enp*, which is written in C++. The kern export and visual patch interface are implemented on the PWGL side and are written in Lisp and CLOS. The connection between the visual PWGL boxes and the Humdrum Tools is realized with the help of the PWGL Shell library, which provides for a graphical interface between PWGL and the unix command-line. Practically every unix command can be turned into a PWGL box and seamlessly used as a part of a patch along with the built-in boxes. When the Humdrum library is loaded, PWGL scans a specific directory defined by the user using the Humdrum library preferences for available Humdrum commands and creates the box interface automatically.

2.1 Kern import/export

Communication between PWGL and Humdrum tools is facilitated by two utilities that convert between ENP and Humdrum data. PWGL has a set of built-in Lisp methods that convert ENP scores into Humdrum data streams or files. Humdrum Extras reverses the process with a program called *hum2enp* to revert back to ENP data. These two converters allow for round-trip processing between ENP objects and Humdrum tools. For example, music generated algorithmically in PWGL can be exported into Humdrum data, then processed with Humdrum tools and converted back into ENP with some analytic markup.

The Humdrum and ENP representations of music are different in several ways. The Humdrum data paradigm is a spreadsheet. Each column (or more generally, a *spine* in Humdrum terminology) of data represents a stream of information such as a part, voice, dynamic, text or analytic data sequence. Each row in the data represents a simultaneity, meaning that all events in the row occur at the same time, with time progressing downwards in the file; any row above the current one occurs in its past, and rows below the current one occur in its future. In other words, horizontal lines of data can be thought of as being the total sounding harmony at any give point in time. This arrangement directly represents the configuration of notes in a musical score, but rotated 90° clockwise and without system breaks as shown in Figure 1. This also means that the lowest part in a file is the left-most spine in the data.

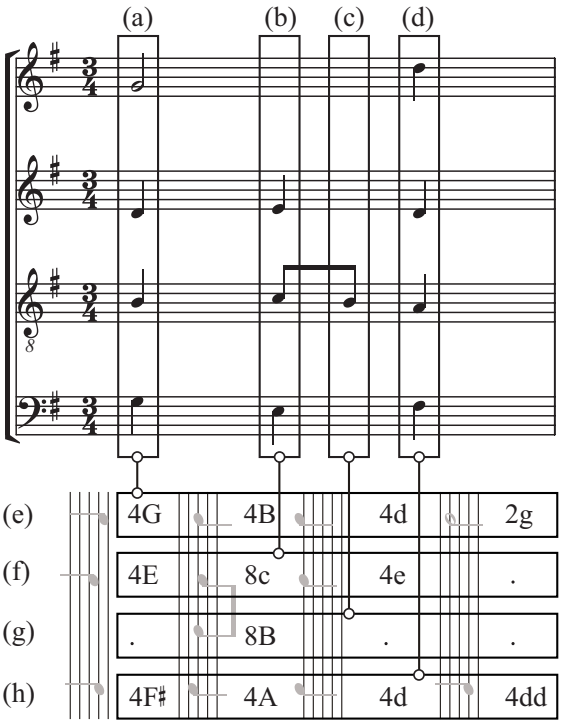
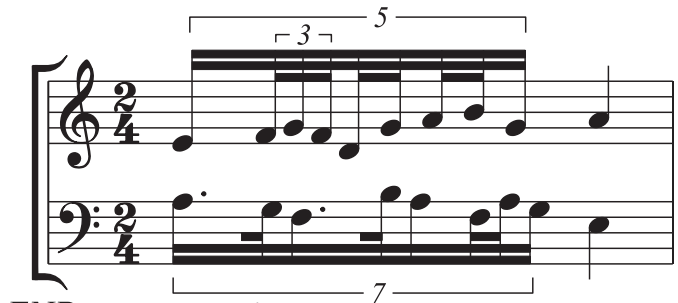


Figure 1. An ENP score displayed as graphical music notation (top) partitioned into harmonic slices (a–d) which are converted into kern records (e–h).

An ENP data file serializes the score by part from highest to lowest in the musical system with a configuration analogous to Type-1 Standard MIDI Files. While Humdrum data is arranged two-dimensionally, ENP data is stored in a hierarchical tree structure by part, voice, rhythm, chord, then note in a similar manner to most XML formats used for encoding musical data. Analytic data is typically stored as *expressions* on the chord or note level. Humdrum files, on the other hand, typically store analytic data in spines (columns) that are parallel to the main streams of musical data.

In order to convert from ENP scores into Humdrum, the scores are first partitioned into *harmonic slices* which effectively represent rows of Humdrum data (see Figure 1). A harmonic slice is created whenever there is at least one note object with a unique start time. Notes sharing start times are collected within the same harmonic slice using an identical process for harmonic analysis in PWGL, or to generate a notated score. Now the harmonic slices, which are already ordered by time, can be written as Humdrum data rows by reading each harmonic slice from bottom to top. Each harmonic slice thus becomes a Humdrum record where the lowest voice is encoded to the left-most column and the highest part in the right-most column. In general, every ENP voice becomes a kern spine within the Humdrum data. The Humdrum export is mainly intended for music analytic applications and therefore the complete ENP score structure is not necessarily preserved. In addition to rhythm and pitch, other notational properties exported from ENP to kern include instrument names, clefs,



ENP representation:

```
(((((1 ((1 :NOTES (64))
       (1 ((1 :NOTES (65)) (1 :NOTES (67)) (1 :NOTES (65))))
       (1 ((1 :NOTES (62)) (1 :NOTES (67))))
       (1 ((1 :NOTES (69)) (1 :NOTES (71))))
       (1 :NOTES (67))))
   (1 ((1 :NOTES (69)))))))
 (((((1
    ((2 ((3 :NOTES (57)) (1 :NOTES (55))))
     (2 ((3 :NOTES (53)) (1 :NOTES (59))))
     (3 ((1 :NOTES (57))
         (1 ((1 :NOTES (53))
             (1 :NOTES (57))))
         (1 :NOTES (55))))))
   (1 ((1 :NOTES (52)))))))
 :STAFF :BASS-STAFF))
```

Humdrum representation:

**kern	**kern
*clefF4	*clefG2
*M2/4	*M2/4
!LO:TUP:r=4:t=7	!LO:TUP:r=4:t=5
28.A	20e
!	!LO:TUP:r=20:t=3
.	60f
56G	.
.	60g
28.F	.
.	60f
.	40d
56B	40g
28A	.
.	40a
.	40b
56F	.
56A	.
.	20g
28G	.
4E	4a
=	=
*-	*-

Figure 2. Sample representation of complex rhythms in ENP and kern data formats.

tempos, time signatures, articulations, and optionally beam and tuplet groupings.

Once the musical data has been transformed into the Humdrum format, it can be processed with Humdrum tools and then converted back into ENP using the *hum2enp* program provided by Humdrum Extras. Humdrum data may contain separate spines for music and analytic data. As the *hum2enp* program converts the musical data into ENP, it collates analytic data into ENP expressions attached to notes or chords.

In order to correctly rebuild the rhythmic hierarchy in ENP, layout extensions have been created for kern data to demark tuplet groupings of arbitrary rhythmic complexity. Figure 2 shows ENP and Humdrum music encodings that represent the same rhythmic formations: a nested quintuplet and an irregularly subdivided septuplet. The structure of ENP is encoded within the parentheses levels starting with the outer pair for the score, then parts, voices, and measures. Rhythmic values are stored within measures as proportions in a hierarchical manner that allows nested tuplet rhythms as shown in Figure 2, where the triplet encompasses a duration of one quintuplet sixteenth note (1/3rd of 1/5th of a beat). The top level of the rhythmic structure is the beat, with subdivisions of the beat inserted as a list of proportions and notes which occur within the duration of their parent rhythmic proportion. Humdrum's linear representation of the music is more direct to the graphical music notation. Each part is a spine (column) of kern data with time progressing downwards. While ENP uses MIDI note numbers to encode pitch (with optional note attributes to resolve enharmonic spellings), kern data uses letter names for pitches. Upper case letters are for the octave below middle C, and lower case for the middle-C octave. Rhythmic values are numbers indicating the note count of that duration which are needed to create a whole note. For example, a septuplet sixteenth note is represented by the number 28, since 28 notes with this duration are needed to create a whole note. Dots following rhythmic values are augmentation dots, so "28." is a dotted septuplet sixteenth note, adding one-half of the dotless duration to the note. Dots in isolation are sequential alignment place holders indicating that the current part has no new activity while something new is occurring in the other part. The rhythmic value 60 can be interpreted as $4 \times 5 \times 3$ (a quarter note divided into 5, then each fifth divided into 3) or $4 \times 3 \times 5$ (a quarter note divided into 3, further divided into 5). The Humdrum representation in Figure 2 shows how to disambiguate between two such possible tuplet interpretations by using optional layout codes for tuplet groupings. The `r` parameter for tuplet layouts indicates the rhythmic duration of the tuplet, and the `t` parameter indicates the tuplet type. Thus the layout instruction `!LO:TUP:r=20:t=3` means that starting on the next note in the data stream (`60f`), a triplet (`t=3`) is to be applied with a total duration of a quintuplet sixteenth note (`r=20`).

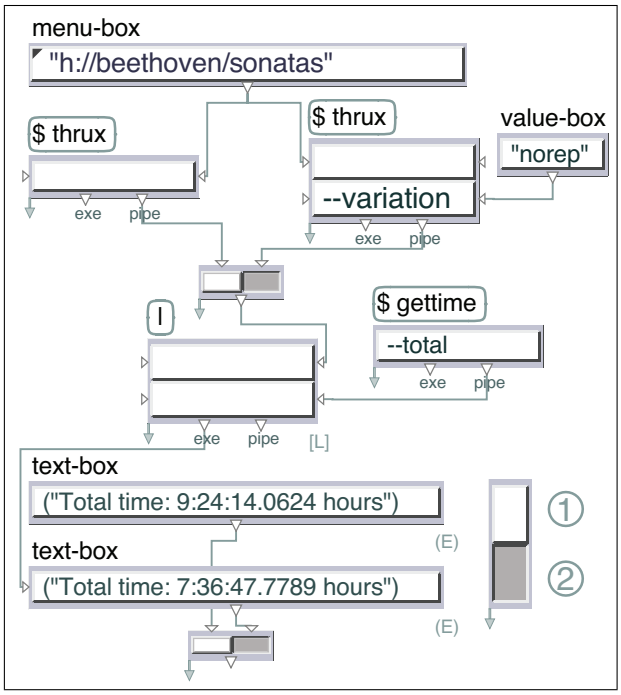


Figure 3. A PWGL Patch containing Humdrum boxes such as *thrux* that are distinguished from regular PWGL boxes by titles enclosed in rounded rectangles.

2.2 Interfacing to Humdrum tools

In addition to data musical transfer between the Humdrum and ENP representational systems, PWGL incorporates access to command-line Humdrum tools as mentioned in the introduction that allows processing of Humdrum data directly within PWGL. Figure 3 illustrates a PWGL patch that interfaces with two Humdrum Extras tools, *thrux* and *gettime*, to measure the total performance duration of a Humdrum data source (either a single work or a stream of multiple works or movements). The patch can be used to calculate the performance duration of input Humdrum data in two configurations: either the performance time when all repeats are taken, or the total amount of time when only second endings are taken. In this case the analysis shows (1) that the complete Beethoven piano sonatas would take about 9.5 hours to perform when taking all repeats (at the tempo specified within the data, taking no breaks between movements), or (2) about 7.5 hours without repeats. The master switch at the bottom right of the patch is used to choose between these two analysis methods to calculate the total performance time. Users of the patch can easily select a different repertory at the top of the patch, and the patch can be displayed in presentation mode to further simplify the interface (see Figures 4 & 5).

The menu-box object at the top of the patch stores a list of data sources, with the complete Beethoven piano sonatas currently selected. In this example the "h://" prefix indicates that the data will be downloaded dynamically from the Internet. [3] When the switch button in the patch

[3] specifically, the h://beethoven/sonatas URI maps onto the URL http://kern.ccarh.org/data?l=beethoven/sonatas

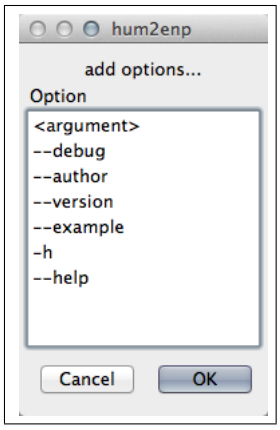


Figure 6. The options dialog for `hum2enp` command showing all the available flags.

causes data to flow through the left-hand *thrux* box, the input data stream is converted from printed ordering of the notation into performance ordering. The right-hand *thrux* box adds the `--variation norep` option to the command, which instructs *thrux* to take only second endings in the music to avoid repetitions. Output from both *thrux* commands is then piped to the *gettime* program. The `--total` option tells *gettime* to calculate the total duration of its input data. Without this option the program calculates the performance time of each harmonic slice. As a comparison, the patch can be expressed in unix shell syntax as two separate commands:

```
thrux h://beethoven/sonatas | gettime -T
thrux h://beethoven/sonatas -v norep | gettime -T
```

where `-v` is an alias for `--variation`, and `-T` for `--total`.

A more elaborate visualization of Humdrum analysis data can be seen in Figures 4 & 5 which display processed Humdrum data in two very different graphic formats on the patch. A menu-box at the top of the patch in Figure 4 is used to select a Humdrum file either from the local hard disk or downloaded via the web. The contents of this file can be pulled through two data paths in the patch. Humdrum data can be transformed into ENP data using the import-kern box and then displayed as graphical music notation within the Score-Editor box. A second path travels through the mkeyscape box which converts the Humdrum data into a graphic image. This image shows a keyscape plot [13] that summarizes the harmonic structure of the chorale, which is primarily in G major with transient key/chord regions on V, vi, and iii. Future work will implement a playback cursor in image boxes which link to the playback position when playing music in Score-Editor boxes. This will allow interactive visual feedback when playing music in the Score-Editor.

All Humdrum Extras command-line tools possess an option called `--options` that lists all options the program recognizes. The PWGL Humdrum library takes advantage of this self-documenting feature to extract a list of options that the program understands. When double-clicking on Humdrum boxes in a PWGL patch, a list of

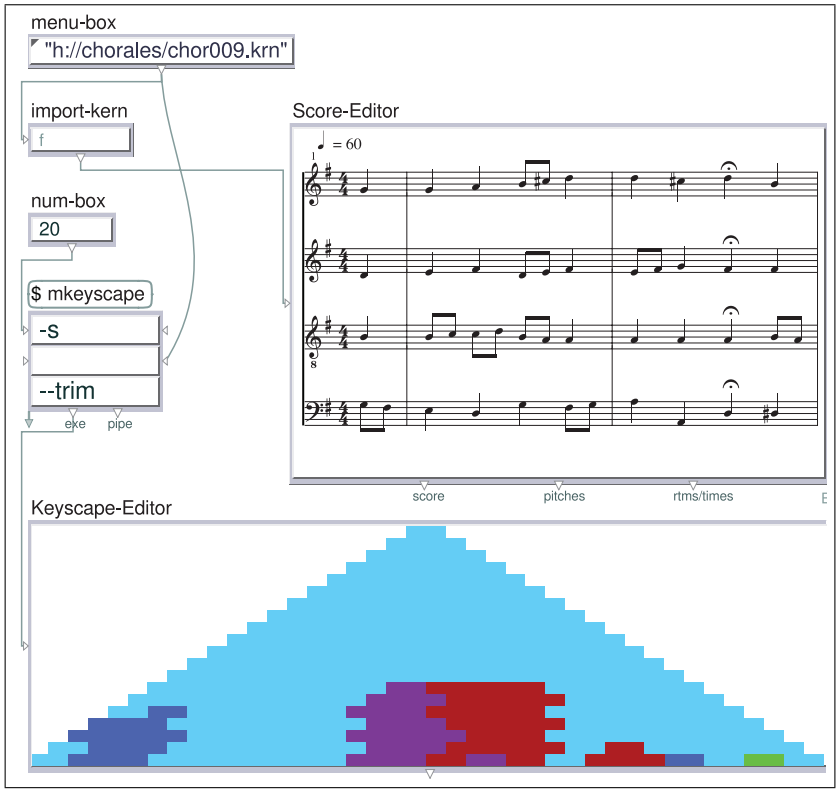


Figure 4. A PWGL patch which loads a Humdrum data file for a J.S. Bach chorale into the ENP Score-Editor as well as a visual analysis of the harmonic structure generated by the Humdrum Extras *mkeyscape* tool.

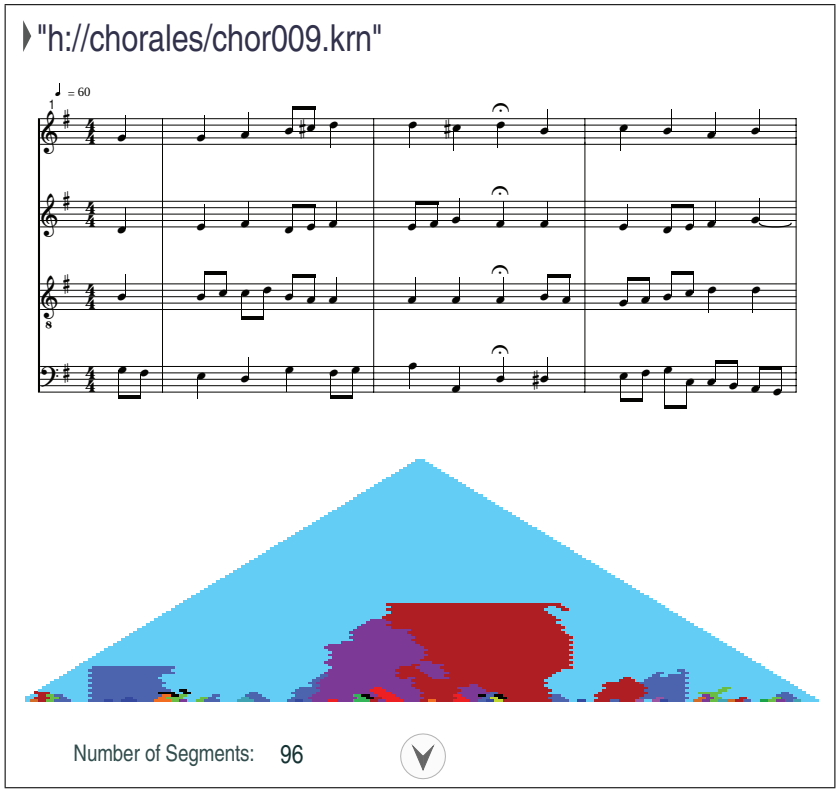


Figure 5. The patch shown in Figure 4 displayed in the presentation mode. The presentation mode allows us to hide the programming details from the end users. It also allows us to lay out the patch in a different way to create a more functional user-interface.

available command-line options appears, such as the list shown in Figure 6. Options can also be defined to accept strings or numbers as arguments, and any default value will also be displayed in the output of the program when using the `--options` option. In this example the `--help` option includes `-h` as a shorter alias.

Figure 5 shows the same patch as in Figure 4, but in *presentation mode*. The PWGL presentation mode is used to hide programming details in a patch to display it in a simplified form that focuses on inputs and outputs, suppressing intermediate details. In presentation mode the connections between boxes are hidden, and individual boxes can themselves be hidden, resized, displaced, as well as scaled or repositioned from their usual programming mode layout. The `-s` option for *mkeyscape*, which controls the number of analysis segments in the triangular image, is visible in Figure 4, taking a value of 20 from the attached num-box. In Figure 5 the call to the *mkeyscape* program is hidden, with only the number of analysis segments visible on the patch. In presentation mode the source score can be selected at the top of the patch, and the number of segments can be changed by typing a new value in place of 96. To recalculate the patch after changing the settings, the arrow at the bottom of the presentation patch is clicked. The program/presentation mode state is persistent for a patch so that it can be saved and then opened in presentation mode. The user can toggle back to the programming mode where visibility and size of all patch boxes are restored as illustrated in Figure 4.

3. CONCLUSIONS

PWGL's Humdrum library enables users of either PWGL or Humdrum to have access to tools available in the other system. For PWGL users the main benefit is access to thousands of classical music scores in the Humdrum file format, while Humdrum users have access to ENP's score editor that provides a rich palette of analytic markup for music notation. The PWGL/Humdrum interface allows users to access the full functionality of the Humdrum toolkit within PWGL and also allows for data exchange between the two systems via the Humdrum file format. The PWGL Humdrum library also offers enhancements over the standard Humdrum Toolkit by providing Humdrum users with a visual interface to Humdrum command-line tools. This is especially beneficial for non-programmers who usually feel more comfortable in a graphical programming environment. Since PWGL is composed of a generalized programming environment as well as the ENP notation editor, data obtained via Humdrum files/tools can be further processed using either pre-existing or user-defined tools within PWGL.

The PWGL Humdrum library can be downloaded at `http://www2.siba.fi/PWGL/downloads`. The `hum2enp` command-line tool is distributed as a part of Humdrum Extras and can be obtained from `http://github.com/craigsapp/humextra`.

4. ACKNOWLEDGMENTS

The work of Mika Kuuskankare has been supported by the Academy of Finland (SA137619). The authors would also like to thank both CCRMA and CCARH at Stanford University for hosting this research.

5. REFERENCES

[1] David Huron. Music information processing using the Humdrum Toolkit: Concepts, examples, and lessons. *Computer Music Journal*, 26(2):15–30, 2002.

[2] Ian Knopke. The Perlhumdrum and Perllilypond toolkits for symbolic music information retrieval. In *International Symposium on Music Information Retrieval*, pages 147–152, 2008.

[3] Andreas Kornstädt. The JRing system for computer-assisted musicological analysis. In *ISMIR*, 2001.

[4] Mika Kuuskankare and Mikael Laurson. Intelligent Scripting in ENP using PWConstraints. In *Proceedings of International Computer Music Conference*, pages 684–687, Miami, USA, 2004.

[5] Mika Kuuskankare and Mikael Laurson. Expressive Notation Package. *Computer Music Journal*, 30(4):67–79, 2006.

[6] Mika Kuuskankare and Mikael Laurson. Survey of music analysis and visualization tools in PWGL. In *Proceedings of International Computer Music Conference*, pages 372–375, 2008.

[7] Mikael Laurson and Mika Kuuskankare. A constraint based approach to musical textures and instrumental writing. In *CP01 workshop on Musical Constraints*, Cyprus, 2001.

[8] Mikael Laurson, Mika Kuuskankare, and Kimmo Kuitunen. Introduction to computer-assited music analysis in PWGL. In *Sound and Music Computing '05*, 2005.

[9] Mikael Laurson, Mika Kuuskankare, and Kimmo Kuitunen. The Visualisation of Computer-assisted Music Analysis Information in PWGL. *Journal of New Music Research*, 37(1):61–76, 2008.

[10] Mikael Laurson, Mika Kuuskankare, and Vesa Norilo. An Overview of PWGL, a Visual Programming Environment for Music. *Computer Music Journal*, 33(1):19–31, 2009.

[11] Vesa Norilo. Introducing Kronos - A Novel Approach to Signal Processing Languages. In Frank Neumann and Victor Lazzarini, editors, *Proceedings of the Linux Audio Conference*, pages 9–16, Maynooth, Ireland, 2011. NUIM.

[12] Craig Stuart Sapp. Online database of scores in the Humdrum file format. In *Proceedings of ISMIR*, pages 664–665, 2005.

[13] Craig Stuart Sapp. *Computational Methods for the Analysis of Musical Structure*. Stanford University, 2011.