DENM (DYNAMIC ENVIRONMENTAL NOTATION FOR MUSIC): INTRODUCING A PERFORMANCE-CENTRIC MUSIC NOTATION INTERFACE

James Bean
University of California, San Diego
jsbean@ucsd.edu

ABSTRACT

In this paper, I describe the state of development for an automatic music notation generator and tablet-based graphical user interface. The programs currently available for the automatic generation of music notation are focused on the compositional and theoretical aspects of the music-making process. **denm** (dynamic environmental notation for music) is being designed to provide tools for the rehearsal and performance of contemporary music. All of the strategies underlying these tools are utilized by performers today. These strategies traditionally involve the re-notation of aspects of a musical score by hand, the process of which can be detrimentally time-consuming. Much of what performers re-notate into their parts is composed of information latent in the musical model—the musical model which is already being represented graphically as the musical score. **denm** will provide this latent information instantaneously to performers with a real-time music notation generator.

1. BACKGROUND

Commercial music typesetting software, such as Finale and Sibelius, are the most common tools for creating musical scores, which require a musician to manually enter musical information via graphical user interfaces. There are cases, for example when compositions are algorithmically generated, where the process of manually entering musical information in this manner is inefficient. As such, programs have been designed to create musical scores where the input from the user is text-based, generated by algorithmic processes, or extracted from spectral analyses.

Most Automatic Notation Generators (ANGs) [1] create a static image, either to be read by musicians from paper, or from a screen displaying it in a Portable Document Format (PDF) representation. LilyPond [2] and GUIDO [3] and convert textual descriptions of music into musical scores. Abjad [4] and FOMUS [5] generate musical score information that can be graphically rendered by LilyPond. OpenMusic [6], Bach [7], PWGL [8] / ENP [9], and JMSL / JSCORE [10] are software tools for composers that generate music notation as part of the compositional process. Belle, Bonne, Sage [11] is a vector graphics library for music notation that enables the drawing of advanced notational concepts not offered by traditional music typesetters. Music21 [12] is a music analysis program that creates score information to be graphically rendered in LilyPond. Spectmore [1] maps spectral analysis information onto a musical score. A few newer ANGs, such as INScore [13] and LiveScore [14], generate animated musical notation for screen representation.

Thus far, ANGs generate static scores that are useful to composers and theorists, and animated scores that are useful for those performing in real-time (described as the *immanent* screen score paradigm by Hope and Vickory [15]). No ANGs specifically target the rehearsal processes of contemporary music performers (a process described as *interpretive* by Hope and Vickory).

I have found that the most critical period for the success of my own works is the rehearsal processes with performers. Performers spend a considerable amount of time in individual rehearsal and group rehearsal settings, and have developed extensive strategies to comprehend, embody, and execute the propositions of composers (see: [16], [17], [18], [19]). Many of the cues that performers notate into their parts are composed of information latent in the musical model—the musical model which is already being represented graphically as the musical score.

denm is software written for iOS devices in the Swift language using the Core Frameworks that enables performers to reap the benefits of these rehearsal strategies without the high cost normally associated with preparing them. Both the musical model and graphical rendering engine are built from scratch to best utilize the touch interfaces of tablet computers. The initial development of **denm** began in the

Javascript language to control the graphical output of Adobe Illustrator. This first phase of development served as research into the systematic organization of musical scores and the programmatic drawing of musical symbols. The vision of this project necessitates animated graphical content, which ultimately required the rewriting of all source code. There are certain features that were prioritized in the initial phase of development [1] that will ultimately be rewritten in an animated context.

2. GRAPHICAL USER INTERFACE

More and more performers are reading music from tablet computers. Software applications like ForScore display a PDF representation of a musical score, allowing a performer to turn pages with a Bluetooth footpedal, as well as to annotate scores with handwritten or typed cues. Performers are able to store many scores on a single device, simplifying the logistics of performing many pieces. Because the PDF contains no reference between graphical musical symbols and their musical functions, the degree to which a player is able to interact with this medium in a musical context is limited.

Many of the cues that performers handwrite in their parts are simplified versions of other players' parts [20]. These types of cues are being reentered by the performer, even though this information is already retrievable from the data that is being graphically represented by the score. The primary objective of **denm** is to expose the structures underlying the music to performers with little cost of access.

2.1 Graphic Design Priorities

The graphic design style of **denm** is minimalist, with as few graphical ornaments as possible. Rather, variations in color, opacity, line-thickness, and other graphical attributes are used to differentiate an object from its environment. In some cases, the variations in graphical attributes serve to differentiate an object's current state from its other potential states. Basic musical symbols, such as clefs and accidentals, have been redesigned to implement this universal design philosophy.

Many of the design choices of standard music notation generators are made with printing in mind. The choices made in **denm** are optimized for display on a screen. The use of thin lines and color is problematic for printers to represent, though these techniques are quite successful with high quality displays.

Figure 1. Design of clefs: treble, bass, alto, tenor.

2.1.1 Clef Design

Traditional clefs take up a considerable amount of horizontal space. The width of traditional clefs is problematic for the spacing of music, particularly when the preservation of proportionate music spacing is a high priority. The minimalist clefs in Fig. 1 take up very little horizontal space. Clefs are colored specifically to enable a differentiation of the clef from the surrounding context and subtle breaks are made in the the staff lines to accentuate the clefs' presence. Staff lines are gray, rather than black, enabling the creation of a foreground / background relationship between musical information carrying objects (notes, accidentals, articulations, etc.) and their parent graph.

2.1.2 Accidental Design

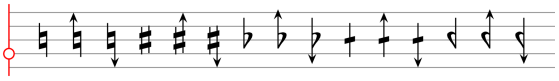

Figure 2. Design of accidentals.

Accidentals, as can be seen in Fig. 2, are drawn programmatically, as opposed to being instances of glyphs from a font. The advantage to uniquely drawing each accidental is that small vertical adjustments can be made to individual components of the object (e.g. body, column(s), arrow) in order to avoid collisions in a more dynamic fashion than is usually implemented in other music notation software [2]. Burnson's work with collision detection of musical symbols [22] serves as an example for the similar work to be approached in continued development.

2.1.3 Rhythm Design

In cases of embedded tuplets, beams are colored by the events' depth in the metrical hierarchy. Ligatures, as seen in Fig. 3, connect tuplet brackets to their events to clarify jumps in depth.

[1] Automatically generated woodwind fingering diagram, string tablature to staff pitch notation conversion, and automatically generated cues.

[2] The initial development of **denm** in Adobe Illustrator-targeted Javascript prioritized this dynamic accidental collision avoidance. Extending the traditional process of avoiding of accidental collisions by stacking accidentals in multiple vertical columns [21], individual adjustments are made to the graphics of the accidentals themselves. Many accidental collisions that traditionally warrant horizontal movement of the objects can be avoided with a single or several small adjustments to individual components of each accidental. Avoiding unnecessary horizontal movement of accidentals makes retaining proportionate music spacing more feasible. More rigorous study of the effects of readability of slightly adjusted accidental graphics is to be undertaken throughout the near-term development of **denm**.

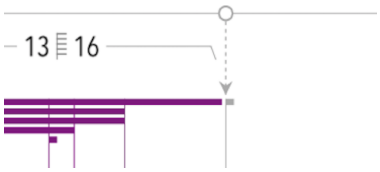

Figure 3. Design of beams and tuplet bracket ligatures

Small graphics, as seen in Fig. 4, indicate the subdivision value that clarify the values of a tuplet. The style of the straight beamlets in the tuplet bracket subdivision graphics mirror the straight beams of rhythms, without the visual noise of traditional flags. Further, the line-thicknesses of beams in the graphics are inversely proportional to their subdivision value, aiding in their visual differentiation. Left edges of tuplet brackets are straight, while right edges of tuplet brackets are angled.

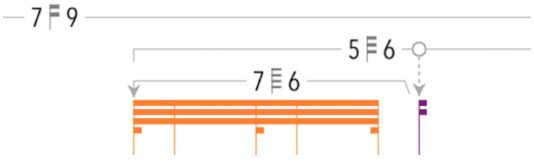

Figure 4. Design of tuplet bracket label graphics

2.2 User Interaction Design Priorities

Many cues that performers notate into their scores are useful at certain points of the learning and rehearsal process, but become less useful at different points in the process. The user interaction design style of **denm** enables performers to determine what musical information is displayed at any point. Performers touch the screen score directly to show or hide certain objects. More advanced user interface strategies will be developed as the underlying analytical procedures (some of them seen in Sec. 3) are implemented.

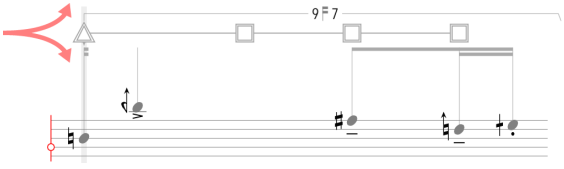

Figure 5. Screenshot of Metronome Graphic revelation.

For example, when a user decides to show a stratum of Metronome Graphics (described further in Sec. 2.3.2), as can be seen in Fig. 5, the entire page recalculates its dimensions, to ensure that the Metronome Graphics take up only the space that they need to. When the user decides to hide that stratum of Metronome Graphics, the layout is recalculated once again, the Metronome Graphics are hidden, and the space they were occupying disappears.

The layout of **denm** is organized as a hierarchy of embedded boxes that recalculate their heights based on what the user elects to show or hide within them. Fig. 6 shows these vertically accumulating boxes. Each box defines its own padding, keeping layout separation consistent for each object.

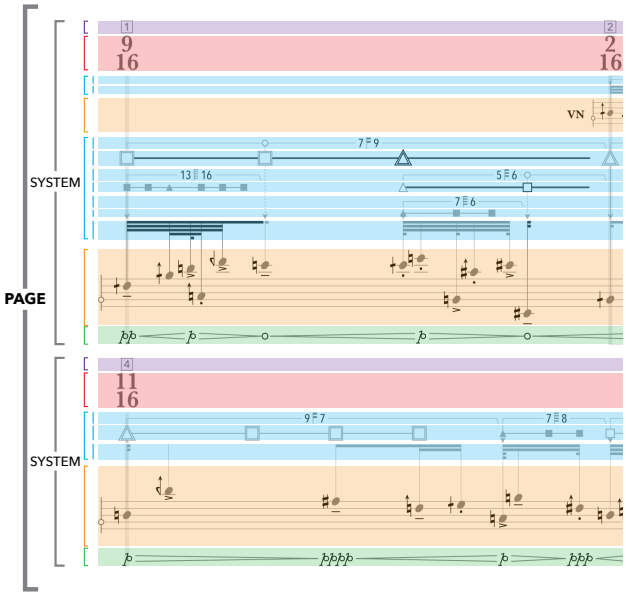

Figure 6. Layout Organization.

2.3 Rhythm Features

2.3.1 Metrical Grid

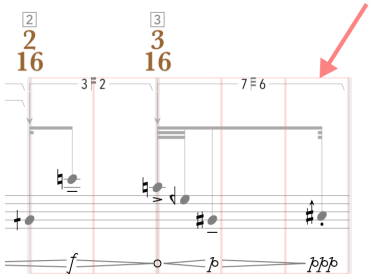

Figure 7. Screenshot of a Metrical Grid.

A performer can tap the time signature of any measure to reveal a grid showing the beats of that measure. This provides a quick reference for the relationship of complex rhythmic events to a global tactus. Quickly drawing lines at the point of each beat in a measure is often the first thing a performer does when receiving a new piece of rhythmically complex music [16], [20].

2.3.2 Metronome Graphics

When a user taps any point in a rhythm, graphics are displayed indicating the best way to subdivide a rhythm (reduced to sequences of duple- and triple-beats). Duple-beats

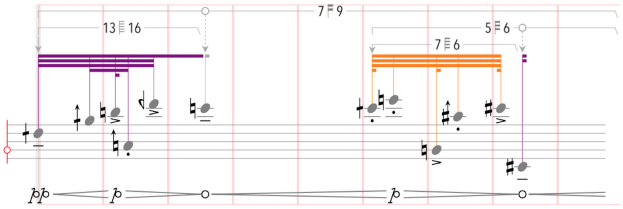

Figure 8. Screenshot of another Metrical Grid.

are represented as rectangles and triple-beats are represented as triangles. Each subdivision-level (e.g. 8th, 16th, 32nd, etc.) has its own graphic, which is a uniquely styled version of the duple- and triple-beat primitives, making the subdivision-level of the metronome understandable at a glance. The process of generating these subdivision references can be seen in Sec. 3.1

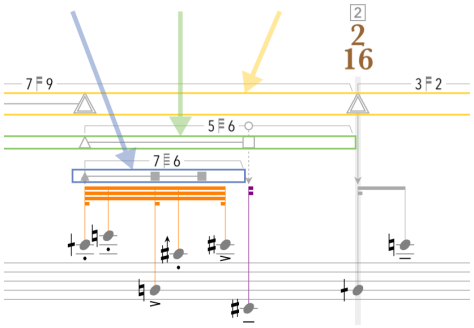

Figure 9. Screenshot of Metronome Graphics.

2.3.3 Metronome Visual Playback

Performers often create click-tracks for learning, rehearsing, and performing rhythmically complex music. Currently, the Metronome Graphic objects can be played-back when a performer taps on the time signature for a measure. The Metronome Graphics "click" by flashing a different color in time. An animated bar progresses from left to right at the speed prescribed by the current tempo of the music. This process has yet to be implemented with other objects in the system, though this will continue to be developed.

As development continues further, a performer will be able to extract any portion of the musical part and rehearse it with the visual click-track of the Metronome Graphics at any tempo. Ultimately, an audio element will be integrated into this metronome process, with sonic attributes mirroring those of the visual Metronome Graphics, to represent subdivision-level and placement in the Metrical Analysis hierarchy (as described in Sec. 3.1).

2.4 Other Players' Parts

Performers often notate aspects of the parts of the other players in an ensemble context. Because this information already exists in the musical model, it can be graphically

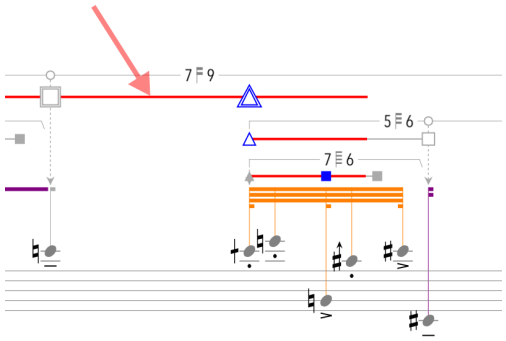

Figure 10. Screenshot of metronome playback.

represented immediately. This feature is currently implemented at a proof-of-concept level. Fig. 11 shows the process of verifying the automatic layout recalculation needed when inserting new musical material. In this case, hard-coded musical material is inserted into the layout when a measure number is tapped by a user.

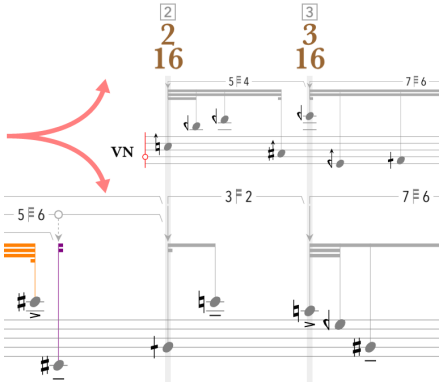

Figure 11. Screenshot of cue revelation.

3. MUSIC ANALYSIS ALGORITHMS

In order to provide performers with rehearsal tools in real-time, robust analysis tools must be developed.

3.1 Metrical Analysis

denm analyzes rhythms of any complexity. The result of this analysis is an optimal manner in which to subdivide the rhythm. Information like syncopation and agogic placement of events can be ascertained from this process. This process can be seen in Alg. 1.

Rhythm in **denm** is modeled hierarchically. The base object in this model is the DurationNode. Any DurationNode that contains children nodes (e.g. traditional single-depth rhythm, or any container in an embedded tuplet) can be analyzed rhythmically. The result of this analysis of a single container node is a MetricalAnalysisNode (a DurationNode itself with leaves strictly containing only duple- or triple-beat durations). MetricalAnalysisNodes are the model used

by the Metronome Graphics, the graphical representation of which is described in Sec. 2.3.2.

Algorithm 1 Metrical Analysis

```
durNodes ←DurationNode.children
parent ←Root MetricalAnalysisNode
function ANALYZE(durNodes, parent)
    s ← durNodes.sum()
    if s = 1 then
        child ←MANode(beats: 2)
        ▷ subdivision level ∗ = 2
        ▷ add child to parent
    else if s <= 3 then
        child ←MANode(beats: s)
        ▷ add child to parent
    else if 4 <= s <= 7 then
        p ← prototypeWithLeastSyncopation
        for pp in p do
            child ←MANode(beats: pp)
            ▷ add child to parent
        end for
    else if 8 <= s <= 9 then
        p ← prototypeWithLeastSyncopation
        if p contains values > 3 then
            for pp in p do
                part ← durNodes partitioned at pp
                newParent ←MANode(beats: cc)
                analyze(part, newParent)
            end for
        end if
    else
        ▷ create array of all combinations
        ▷ of values 4 <= v <= 7 with sum of s
        c ← combinationWithLeastSyncopation
        for cc in c do
            part ← durNodes partitioned at cc
            newParent ←MANode(beats: cc)
            analyze(part, newParent)
        end for
    end if
end function
```

A MetricalAnalysisPrototype is a sequence of two or three elements, each of which having a duple- or triple-beat duration. Values between and including 4 and 7 have a dedicated list of prototypes [3] , each of which can be compared against any rhythm with a relative durational sum of the same value, the process of which can be seen in Alg. 2.

Algorithm 2 Syncopation

```
d ← durationNodes.cumulative()
▷ e.g. [4, 5, 7] ← [4, 1, 2].cumulative()
p ← prototype.cumulative()
▷ e.g. [2, 4, 7] ← [2, 2, 3].cumulative()
syncopation ← 0
function GETSYNCOPATION(d, p)
    if d[0] = p[0] then
        ▷ Rhythm beat falls on prototype beat
        ▷ No syncopation penalty added
        ▷ Adjust d and s accordingly
        getSyncopation(d, s)
    else if d[0] < p[0] then
        ▷ Rhythm beat falls before prototype beat
        ▷ Check if next duration falls on prototype beat
        if delayedMatch then
            ▷ No syncopation penalty added
        else
            syncopation ← syncopation + penalty
        end if
        ▷ Adjust d and s accordingly
        getSyncopation(d, s)
    else
        ▷ Rhythm beat falls after prototype beat
        ▷ Check if next prototype value falls on duration
        if delayedMatch then
            ▷ No syncopation penalty added
        else
            syncopation ← syncopation + penalty
        end if
        ▷ Adjust d and s accordingly
        getSyncopation(d, s)
    end if
end function
```

Cuthbert and Ariza [12] apply their Metrical Analysis process to beaming in the score representation of rhythms. This strategy will be the model for continued development in **denm**, extending to cases of arbitrarily-deep nested-tuplet rhythms.

[3] List of prototype sequences by relative durational sum:
$4 : (2, 2)$
$5 : (3, 2), (2, 3)$
$6 : (2, 2, 2), (3, 3)$
$7 : (2, 2, 3), (3, 2, 2), (2, 3, 2)$
$8 : (4, 4), (3, 3, 2), (2, 3, 3), (3, 2, 3) *$
$9 : (3, 3, 3), (4, 5), (5, 4) *$
$*$ Rhythms with relative durational sums of 8 and 9 are compared against all of these combinations shown here. If the combination with least syncopation contains only values of 2 or 3, a MetricalAnalysisPrototype is generated with children containing Durations of duple- and triple-values. In the case that the combination with least syncopation contains values > 3, internal MetricalAnalysisNodes are created with the Durations of these values. The original DurationNode array is partitioned at points determined by the combination. The MetricalAnalysis process is then applied for each partition, and MetricalAnalysisNode leaves with duple- and triple-value Durations are added to each of these internal MetricalAnalysisNodes.

3.2 Pitch Spelling

Effective pitch spelling is critical in the process of generating staff notation for music that is algorithmically composed or extracted from spectral analyses. Algorithms for pitch spelling within tonal musical contexts have been compared by Meredith [23] and Kilian [24]. The musical contexts that **denm** is most immediately supporting are rarely tonal. More often these musical contexts are microtonal. The preferences in the current tonally-based pitch spelling algorithms, however, are defined by establishing tonal center.

The benefits of tonal-center-based pitch spelling are lost in atonal musical contexts. Rather, primitive intervallic relationships are preserved objectively, rather than being subjected to the requirements of a tonal center. When spelling pitches in microtonal contexts, other pragmatics arise that influence the decision-making process.

The short-term goal of the pitch spelling process in **denm** is to spell microtonal polyphonic music up to the resolution of an 1/8-tone (48 equal divisions of the octave). In time, this process may be extended to accommodate other tuning systems. The development of this microtonal pitch spelling procedure is in process. Currently, dyads of any combination of resolutions (1/2-tone, 1/4-tone, 1/8-tone) are spelled correctly, though more rigorous testing is underway to verify this. Further development of this pitch spelling algorithm into homophonic and polyphonic microtonal contexts will incorporate aspects of Cambouropoulos' shifting overlapping windowing technique [25].

4. INPUT FORMATS

At this point in development, there is a working prototype input text format. All figures in this paper have been created with this input text format. A long-term priority for development is to build conversion from common music interchange formats, such as musicXML [26], into the native **denm** text input format. This conversion will enable composers to generate and input music in the style that best serves them, while they benefit from a performer-facing interactive graphical user interface.

```
create a new measure ──→ #
create a new rhythmic group ──→ 9,16 FL
relative duration of event ──→     2 --
create an embedded tuplet ──→          4 p 74.5 d pp a -
simply by indenting the next           2 p 79.75
events.                                1 p 83 d p a >
                                       2 p 69.25 a .
                                       4 p 85.5 a >
                                   2 p 84 d o a -
                                   3 --
Embedding can occur to any ──→         3 --
depth.                                     1 p 84.5 a .
                                           2 p 88 d p a .
                                           1 p 67 a >
                                           2 p 82.25 a .
                                           1 p 85 a >
                                       2 p 61 d o a -
```

Figure 12. Demonstration of text input syntax.

5. FUTURE WORK

The current development of **denm** is focused on building robust musical operations within the musical model. The extension of the microtonal pitch spelling procedure into homophonic and polyphonic contexts is in development now. Once the pitch spelling procedure is completed and tested, the accidental collision avoidance procedure will be of primary focus. In the longer-term, development will center around the extension of the musical model into multi-voiced and multi-part musical contexts. When these steps are completed, this more fully-featured musical model will be hooked into the graphical realm. User interface strategies will be developed in accordance with the advancement in the musical model and graphical capabilities.

6. REFERENCES

[1] H. Wulfson, G. D. Barrett, and M. Winter, "Automatic notation generators," in *Proceedings of the 7th International Conference on New Interfaces for Musical Expression*, ser. NIME '07, 2007, pp. 346–351.

[2] H.-W. Nienhuys and N. Jan, "Lilypond, a system for automated music engraving," *Colloquium on Musical Informatics (XIV CIM 2003)*, May 2003.

[3] K. Renz, "Algorithms and data structures for a music notation system based on guido music notation," Ph.D. dissertation.

[4] T. Baca, J. Oberholtzer, and V. Adan. In conversation. [Online]. Available: `http://abjad.mbrsi.org/in_conversation/from_trevor_josiah_and_victor/index.html`

[5] D. Psenicka, "Fomus, a music notation software package for computer music composers," in *International Computer Music Conference*. San Francisco: International Computer Music Association., 2007, pp. 75–78. San Francisco.

[6] J. Bresson, C. Agon, and G. Assayag, "Openmusic: Visual programming environment for music composition, analysis and research," in *Proceedings of the 19th ACM international conference on Multimedia*. ACM, 2011, pp. 743–746.

[7] A. Agostini and D. Ghisi. What is bach? [Online]. Available: http://www.bachproject.net/what-is-bach

[8] M. Laurson, M. Kuuskankare, and V. Norilo, "An overview of pwgl, a visual programming environment for music," *Computer Music Journal*, vol. 33, no. 1, pp. 19–31, March 2009.

[9] M. Kuuskankare and M. Laurson, "Expressive notation package-an overview." in *ISMIR*, 2004.

[10] N. Didkovsky and P. Burk, "Java music specification language, an introduction and overview," in *Proceedings of the International Computer Music Conference*, 2001, pp. 123–126.

[11] W. Burnson, *Introducing Belle, Bonne, Sage*. Ann Arbor, MI: MPublishing, University of Michigan Library, 2010.

[12] M. S. Cuthbert and C. Ariza, "music21: A toolkit for computer-aided musicology and symbolic music data," J. S. Downie and R. C. Veltkamp, Eds., vol. 11. Utrecht, Netherlands: International Society for Music Information Retrieval, August 2010, pp. 637–642.

[13] D. Fober, Y. Orlarey, and S. Letz, "Inscore: An environment for the design of live music scores," *Proceedings of the Linux Audio Conference - LAC 2012*, 2014.

[14] G. D. Barrett and M. Winter, "Livescore: Real-time notation in the music of harris wulfson," *Contemporary Music Review*, vol. 29, no. 1, pp. 55–62, 2010.

[15] C. Hope and L. Vickery, "Visualising the score: Screening scores in realtime performance," 2011.

[16] S. Schick, "Developing an interpretive context: Learning brian ferneyhough's bone alphabet," *Perspectives of new music*, pp. 132–153, 1994.

[17] I. Pace, *Notation, Time and the Performer's Relationship to the Score in Contemporary Music*. Leuven University Press, 2009.

[18] S. E. Kanach, *Performing Xenakis*. Pendragon, 2010.

[19] K. Schulmeister, "The interpretation of new complexity: Reflections on the learning process of [d(ks)b] by joan arnau pàmies," April 2012.

[20] P. Archbold, "Performing complexity."

[21] D. Spreadbury, "Accidental stacking." [Online]. Available: http://blog.steinberg.net/2014/03/development-diary-part-six/

[22] W. A. Burnson, H. G. Kaper, and S. Tipei, *Automatic Notation Of Computer-Generated Scores For Instruments, Voices And Electro-Acoustic Sounds*. Citeseer.

[23] D. Meredith, "Comparing pitch spelling algorithms on a large corpus of tonal music," in *Computer Music Modeling and Retrieval*. Springer, 2005, pp. 173–192.

[24] J. Kilian, "Inferring score level musical information from low-level musical data," Ph.D. dissertation, TU Darmstadt, 2005.

[25] E. Cambouropoulos, "Pitch spelling: A computational model," *Music Perception*, vol. 20, no. 4, pp. 411–429, 2003.

[26] S. Cunningham, "Suitability of MusicXML as a Format for Computer Music Notation and Interchange," in *Proceedings of IADIS Applied Computing 2004 International Conference, Lisbon, Portugal*, 2004.