NEON.JS: NEUME EDITOR ONLINE

Gregory Burlet, Alastair Porter, Andrew Hankinson, Ichiro Fujinaga
Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT)
McGill University, Montréal, Québec, Canada
{gregory.burlet,alastair.porter,andrew.hankinson}@mail.mcgill.ca, ich@music.mcgill.ca

ABSTRACT

This paper introduces Neon.js, a browser-based music notation editor written in JavaScript. The editor can be used to manipulate digitally encoded musical scores in square-note notation. This type of notation presents certain challenges to a music notation editor, since many neumes (groups of pitches) are ligatures—continuous graphical symbols that represent multiple notes. Neon.js will serve as a component within an online optical music recognition framework. The primary purpose of the editor is to provide a readily accessible interface to easily correct errors made in the process of optical music recognition. In this context, we envision an environment that promotes crowdsourcing to further the creation of editable and searchable online symbolic music collections and for generating and editing ground-truth data to train optical music recognition algorithms.

1. INTRODUCTION

Music notation editors allow users to manipulate the arrangement of symbols within a digitally encoded musical score. Given the complexity of some musical works, the arrangement of symbols may have intricate relationships. Consequently, a music notation editor must have knowledge of these symbols and their relationships in a musical context to ensure that transformations to the symbols in the editor yield a music score that is syntactically correct. For example, the editor should ensure that notes are placed correctly on the staff and that musical properties are not violated.

There are three main issues that a music notation editor must address: the notation encoding schema, the rendering of symbols, and the relationship between the notation encoding and the graphical representation. First, the digital representation of symbols in the music notation system must be systematically defined so that it represents the desired musical structures and hierarchies. Second, each symbol must be graphically rendered as glyphs on the screen. Finally, changes made by the user in the graphical interface must be translated accurately and completely to the underlying encoding—a non-trivial task since there may be cascading effects on other symbols in the encoded document.

This paper introduces Neon.js, a music notation editor for an early notation system known as *square-note notation*, which originated in the 13th century. In this system of notation, individual notes may be grouped into larger structures called *neumes*, which represent the pitches of a vocal phrase spanning a single syllable. Neumes have different names that are determined by the pitch contour of the individual notes making up the neume. Most neumes are *ligatures*—symbols that represent multiple connected notes. Square-note notation editors must be able to correctly render these ligatures, while still allowing access to individual notes in the neume. An overview of this notation system and example scores can be found in the *Liber Usualis*, a compilation of plainchant used by the Roman Catholic Church [3].

Neon.js is a web application. It displays square-note notation in the web browser and accepts user input to modify the underlying digital representation of the score. Neon.js can be used to create new musical scores, or to correct errors from automated transcriptions in an optical music recognition (OMR) workflow.

By making the editor easily accessible online, OMR error correction tasks can be distributed amongst many people, potentially accelerating the creation of ground-truth training data and errorless symbolic music collections. The process of outsourcing a simple, defined task to the general population is known as crowdsourcing. We predict that by providing tools to enable crowdsourcing of the correction of OMR transcription errors, early music researchers will be able to obtain transcriptions more quickly in order to perform computer-assisted analyses of these works.

We begin with a review of music notation editors and online crowdsourcing systems. Next, we discuss the benefits of a browser-based editor, the requirements for a musical document-encoding scheme, and available techniques for drawing in a web browser. Finally, the software architecture of Neon.js is introduced and the main functions of the editor are described, with a focus on functionality that is unique to square-note notation.

2. RELATED WORK

2.1 Music Notation Editors

An editor for square-note notation contains some functionality that is similar to editors of common music notation. The core functionality required of a notation editor is to allow the user to insert, delete, and change symbols in a piece of notated music. Well-known music notation editors such as Finale[1], Sibelius[2], MuseScore[3], and NoteFlight[4] provide these basic functions and more. Of these, NoteFlight is the only known example of a web-based editor for common music notation. All of these editors allow the user to save files containing the digital encoding of the edited musical document. These editors allow changes to individual symbol attributes, like note pitch and ornamentation, as well as changes to global musical properties, such as tempo and time signature. Since certain properties of common music notation do not exist in square-note notation, an editor for neume notation does not need to handle features such as tempo, time signature, chords, multiple voices on a staff, beams, slurs, or tuplet marks.

Editors for square-note notation do exist, although they are significantly less prevalent than editors for common music notation. The Medieval plugin[5] for Finale allows square-note notation to be edited in Finale. To input and render this notation in Finale, the plugin must work around some syntactic musical requirements that Finale imposes on scores. For example, the plugin alters the time signature of each bar in the score (each of which may have a different number of notes), to prevent Finale from automatically inserting bar lines.

Another neume notation editor has been developed as part of the NEumed Unicode Manuscript Encoding Standard (NEUMES) project [2]. The editor is developed as a Java applet and is therefore available for use in a web browser[6]. The editor uses NeumesXML, an encoding format that describes square-note notation as well as earlier neume notation systems without staff lines. NeumesXML is an XML-based format that uses Unicode characters to represent different neumes. Although the editor did not advance beyond the prototyping phase of development, the user interface allows scanned images of musical scores to be displayed side-by-side with the rendered notation for reference.

As part of the Tübingen project [7], the MEI-Neumes-Viewer[7] renders neume notation in the web browser. In the preliminary stages of the project, the *Humdrum Toolkit* was used to develop a data representation specific to their repertoire. In later stages of the project[8] they developed the neume-encoding scheme that is now part of the Music Encoding Initiative (MEI). The MEI-Neumes-Viewer is an engraving system, not an editor, and so does not provide the ability to interactively edit a score.

Many music notation encoding schemes have been proposed over the years, most of which are tailored to a specific notation system. However, few formats seek to provide a universal data representation that encompasses and describes all musical notation systems. A music notation editor requires a digital representation of the symbolic notation because it needs to be stored and processed by a computer. Digital encoding systems should be explicit and declarative to prevent loss of information [11]. An encoding system should not require software applications to derive relationships between elements in the musical score. Instead, all relationships should be explicitly described. For neume notation, the digital encoding should preserve neume types and pitch information [7].

Aruspix, an OMR system for recognizing early printed music [10], contains an editor that is used to correct recognition errors in OMR output. Aruspix renders the recognized score over top of the original image, facilitating error detection and correction by the user.

2.2 Crowdsourcing Systems

Many projects have benefited from using online crowdsourcing techniques to harness the computational power of many humans (e.g., Wikipedia). The reCAPTCHA project [1] has successfully transcribed over 440 million words that were unrecognizable by optical character recognition algorithms. This was accomplished by presenting transformed words as “Completely Automated Public Turing test to tell Computers and Humans Apart” (CAPTCHA) challenges on the web. These tests are designed to prevent malicious software from performing actions that should only be performed by humans, who must transcribe the machine-unrecognizable words to prove they are human. In the process, optical character recognition errors are corrected [1]. When this is deployed over millions of websites, it becomes a highly effective method of performing large-scale text correction. A further advantage of online crowdsourcing for document digitization is that it is inexpensive and requires minimal training for the contributors, compared to the cost and training required to set up a scanning and recognition workstation [8].

3. ONLINE EDITING

3.1 Web Versus Desktop Applications

The AJAX programming technique [9] for websites enables the creation of interactive web applications that behave like desktop applications. This has lead to discussions, involving both users and software developers, about the advantages and disadvantages of creating web applications over desktop applications.

[1] http://www.finalemusic.com
[2] http://www.sibelius.com
[3] http://musescore.org
[4] http://www.noteflight.com
[5] http://www.klemm-music.de/notation/medieval
[6] http://www.scribeserver.com/medieval/staves_applet.html
[7] http://www.dimused.uni-tuebingen.de/hildegard
[8] http://www.dimused.uni-tuebingen.de/tuebingen_phase1_e.php

There are many features of web applications that are appealing to application users and developers. Unlike desktop applications, web applications can be continually updated by developers with new features and do not need to be updated by the end-user when a new version is released. Another feature is that any user with an Internet connection and a web browser can access and use the application, whereas desktop applications require time to install and typically have different installation procedures depending on the operating system of the client machine. While not necessarily browser independent, web applications are platform independent, creating a larger user base.

There are, however, disadvantages of hosting a music notation editor online. The most notable disadvantage is that the user must be connected to the Internet to use the application. Compared to desktop applications, web applications that have a client-server architecture can be architecturally complex since interactions and synchronicity between the client and server must be maintained. Advocates of desktop application development also claim that web applications exhibit slower performance and discontinuous user interactions in relation to compiled programs. These deficits are becoming negligible now that contemporary web browsers have become faster at executing JavaScript, and the use of AJAX has enabled seamless and transparent communication between the client and server.

Web applications have a lower barrier to entry than desktop applications. Fewer obstacles are presented to users interested in using the software, making them more suitable for use in crowdsourcing applications targeted to a large number of people.

3.2 Crowdsourcing online

Crowdsourcing may be used in an OMR workflow for the purposes of quickly and inexpensively correcting errors in digitized musical documents. Pattern recognition algorithms are not perfect—there will inevitably be recognition errors that must be fixed by a human. A single person can allocate a large amount of time to perform these corrections, or this unwieldy workload can be distributed amongst many people. The resulting digital encoding can then be archived, indexed for searching, or used as ground-truth data to further train OMR algorithms.

In creating an online crowdsourcing system for the correction of OMR errors, there are four challenges that developers of these systems must consider [4]. First, users must be recruited and their interest maintained. A potential recruitment problem exists for the correction of square-note notation, since early music is less popular than contemporary music. Next, large tasks must be decomposed into manageable sub-problems, which are easily solvable by a single person. This method of problem solving is called divide-and-conquer. For OMR, error correction of an entire corpus can be broken down into smaller tasks where segments of music, such as a single page, line, or bar, are corrected. Each subtask would require users to correct the position and pitch of erroneous notes. A typical correction task is displayed in Figure 1. By providing an image of the original musical document for reference, a task that would normally require domain-specific musical knowledge can be posed as a comparison problem. The task of correcting pitch and position errors involves dragging the incorrectly recognized notes to match the position of those notes on the source image. This increases the number of possible contributors that can be recruited to perform correction tasks. One potential limitation of this correction scheme occurs when the neume type is incorrectly recognized. In this case, the incorrect glyph must be deleted and the correct glyph inserted to match the original score—a task that may require musical knowledge and may be surrendered to another user. The encouragement and retention scheme known as instant gratification could be used to immediately show contributors how their contributions are making a difference [4]. After correcting errors within a segment of music, displaying or auralizing the full score would not only emphasize the contributions of the user, but also encourage future corrections of OMR output.

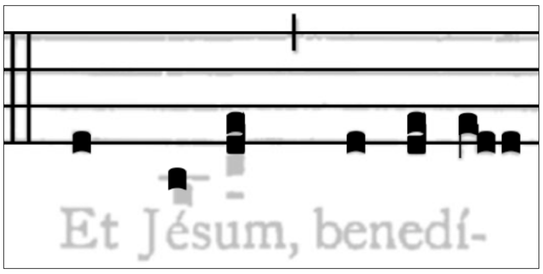

Figure 1. An example of a pitch correction task. Some recognized notes (dark) need to be moved to the same location as on the original score (light).

The last two challenges a crowdsourcing system must consider is the evaluation of user contributions, and the combination of these contributions to solve the target problem. To manage OMR error corrections by multiple contributors, an ideal system might present the same segment of music to more than one person for correction. An automated voting system [1] can be used to choose the most commonly made correction for a segment of music and then combine segments into the final digitized score.

3.3 Tools for Musical Engraving and Interaction

Adobe Flash, Scalable Vector Graphics (SVG), and the HTML canvas element are technologies available for dynamic drawing in the web browser. Flash is a popular web technology that supports interactive manipulation of the drawing surface with scripts to produce interactive

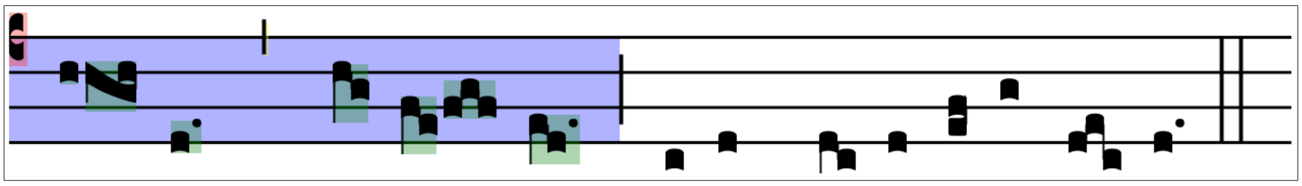

Figure 2. Neon.js rendering one system of the *Liber Usualis*, with and without bounding boxes.

and media-driven web applications. However, web applications that use Flash require an additional program to be installed as a browser plugin in order to be used. Some operating systems and devices do not support Flash and therefore cannot run these applications.

SVG is an XML format for describing images as vectors—mathematical descriptions of lines and curves. This representation enables SVG images to be rendered at any size, making the format a good choice for images that require a high level of detail. Most modern web browsers can display SVG images and manipulate them using JavaScript.

The HTML canvas element is a low-level raster drawing surface supported by modern web browsers, which can be controlled with JavaScript. JavaScript libraries such as jQuery [10] can be used to simplify common tasks such as event handling, animations, and AJAX interactions. The canvas element implements a "fire-and-forget" model, where a drawing surface is presented as a two-dimensional array of pixels that can be independently coloured. Although images drawn on the canvas can be easily manipulated, interaction with the drawings requires some additional overhead. Since the canvas "forgets" what was drawn, the state of objects drawn to the canvas needs to be maintained. When the state of an object changes (e.g., dimensions and colour), the canvas is repainted to reflect these changes.

4. A NEUME NOTATION EDITOR

This section introduces our web-based neume notation editor, Neon.js. We start with a description of music notation encoding and how it can be rendered in a web browser. We then describe how the music notation editor links the notation encoding and the graphical representation.

4.1 Notation Encoding

Neon.js reads and writes files encoded in the Music Encoding Initiative (MEI) format. MEI is an XML-based file format for the representation of many notation formats. The MEI schema is split into a "core", which defines features common to many notation formats, and a set of additional modules that each define a specific notation system [6]. Neon.js uses the Solesmes module, an extension to the MEI core that allows representation of square-note notation along with other specific practices particular to the notation system used by monks in Solesmes, France in the 19th century. These practices include *divisions* (breath marks), *episemata* (note stresses), and unique neume names.

4.2 Notation Rendering

We decided to use the canvas element for rendering scores in Neon.js. We store images of neumes and ligatures as SVG so that the score can be rendered in detail at any zoom level. We use Fabric.js[11], a library that provides high-level drawing functions such as lines and boxes. Fabric.js is also used to render our SVG images directly onto the canvas drawing surface in the browser.

MEI files that have been transcribed by OMR contain the physical locations on the page of each recognized element, stored as a bounding box surrounding that element. We use these physical locations to calculate where to draw the musical symbols, including systems, clefs, neumes, and divisions. An example of Neon.js rendering one system of music from the *Liber Usualis* is shown in Figure 2.

Neon.js draws elements of the score on top of an image of the musical document that the score was transcribed from. The user can adjust the transparency of the background image to show just the rendered notation, or both the background and notation.

4.3 Software Architecture

Neon.js is built using a client-server architecture. The Neon.js client renders the musical score in the browser and transforms user input into edit commands that are sent to the server. We use AJAX to send edit commands from the client to the server without needing to reload the web page. The server receives these commands from the client and applies them to a stored MEI file, saving the changes. The server is written in Tornado[12], an open-source Python web server framework. To read, manipulate, and write MEI files on the server, Neon.js makes use of the Python bindings of libmei[13], an open-source C++ library [6].

The Neon.js client uses object-oriented programming techniques and the model-view-controller design pattern [5] to separate display from musical knowledge. The role

[10] http://jquery.com

[11] http://fabricjs.com
[12] http://www.tornadoweb.org
[13] http://ddmal.music.mcgill.ca/libmei

of the model-view-controller design pattern is to isolate application logic in the model from display and user interface logic in the view through an intermediary controller that coordinates the two. The model keeps the state of all of the musical elements on a score. When a user modifies an element, the score is redrawn to reflect the changes to the object's state. This means that changing the data representation from MEI to NeumesXML, for example, would not affect the drawing code. Similarly, changing the drawing medium from canvas to SVG, for example, would not affect the data representation and editing functionality.

5. EDITING FUNCTIONS

To develop a more thorough understanding of the major functions of Neon.js, the structure of an instantiated neume object will be described. In Neon.js, a neume is represented as a sequence of puncta. A punctum is the simplest type of neume, consisting of only one note that is represented as a single square. Only the note name and octave of the first note is stored in the model. Subsequent notes are encoded as having a pitch relative to the first note. The following client-side functions operate on this information and then call a corresponding server function to update the underlying MEI file. In this section we focus on implementation details of the main editing functions that are specific to neume notation.

5.1 Inserting and Deleting Neumes

The only neume type that can be added to the score through the user interface is the punctum. Neumes that are composed of more than one note are entered by inserting a punctum for each pitch in the neume, then combining them with the neumify function (Section 5.2). This feature lets a user focus on the melodic content of the score without needing to identify the name of each neume to insert.

Neumes may be deleted. When a neume contains multiple notes, all of the notes in the neume are deleted. To delete individual notes within a neume, it must first be ungrouped (Section 5.3).

5.2 Neumify

In the same way that ligatures pose problems for OMR systems by obfuscating pitches of individual notes [12], ligatures within neumes pose problems for square-note notation editors. In many cases, the graphical representation of a neume is not a simple concatenation of the selected glyphs. Figure 3 shows three notes being selected and combined by the neumify function into a *porrectus* neume, identified by the downward then upward melodic contour of the notes. Neon.js needs to be able to recognize a neume by its contour in order to draw it with the correct ligatures. MEI also requires the name of a neume to be encoded along with the notes that make up the neume. The neumify function infers the name of a neume from the pitch differences in a sequence of notes.

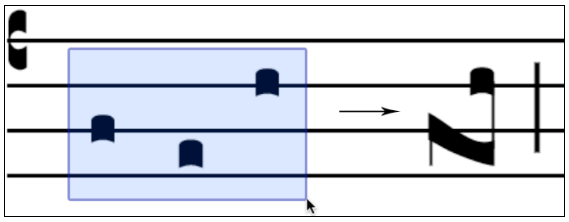

Figure 3. A use case where the user selects a set of puncta and applies the neumify function to create a *porrectus* neume.

When the neumify function processes a sequence of notes, the melodic contour of the notes is calculated. To describe the relationship of a note to its previous note, -1, 0, or 1 is used to represent a lower pitch, the same pitch, or a higher pitch, respectively. We create a prefix tree containing all of the neume types that Neon.js can render. The edges of the tree represent the direction of movement between two notes. For example, the *porrectus* neume from Figure 3 has a contour of -1, 1. The traversal of a partial prefix tree is shown in Figure 4 with bolded lines. Using a prefix tree for lookups means that the speed of identifying a neume type is dependent on the number of notes in the neume and not the size of the tree, which means that more neume types can be added without affecting the speed of lookups. When the melodic movement of a sequence of puncta cannot be matched to a known neume type, the neume type is labeled "compound neume". Instead of naïvely concatenating the selected glyphs to form the drawing of a compound neume, the longest matching prefix in the tree can be used to determine the best way of rendering notes within the neume. Neon.js is currently capable of differentiating between 44 distinct neume types.

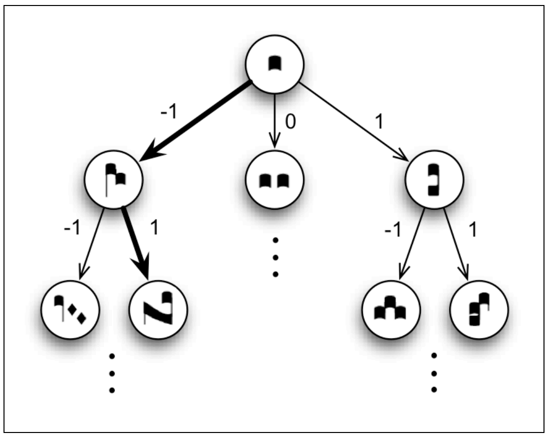

Figure 4. An example of deriving the *porrectus* neume type by searching a prefix tree. The bolded arrows reveal the traversal of the tree.

5.3 Ungroup

The ungroup function in Neon.js performs the inverse of the neumify function. When the ungroup function is applied to a neume, the neume is replaced by puncta having the same pitches as the underlying notes in the neume. This functionality lets a user adjust properties of a single note in a neume, such as pitch and ornamentation, and then regroup the neume with the neumify function.

5.4 Modify Pitch

Neon.js allows the user to modify pitches of neumes through a click and drag interface. The notes automatically snap to spaces or lines within the staff. When the user moves a neume comprised of multiple notes, all notes within the neume are shifted by the same amount.

6. CONCLUSION

Neon.js[14] is an online and open-source neume notation editor developed as a web application. The editor uses the HTML canvas element to render musical symbols in the web browser. Neon.js supports MEI as a digital data representation for square-note notation and uses open-source software libraries to facilitate manipulation of this data format. A minimal but powerful graphical user interface allows the user to digitally replicate or edit early music documents containing square-note notation. As the user modifies the arrangement of musical symbols, the underlying MEI file is modified to reflect these changes.

As a web application, the software is easily accessible, requires no installation, and enables large-scale crowdsourcing to distribute the task of correcting note pitch and position errors made in the process of optical music recognition. We hope that the resulting symbolic music collections will then be indexed and available for search at a single location, creating a substantial source of information for early music researchers. We also hope that this centralized collection of corrected symbolic musical documents is made available for use as ground-truth training data for the digitization of other manuscripts in square-note notation. Although developed as a component of an online optical music recognition workflow, Neon.js can also be used to manually transcribe or write new works.

7. ACKNOWLEDGEMENTS

The authors would like to thank Catherine Motuz for her answers to our early music notation questions. This project has been funded with the generous financial support of the Social Sciences and Humanities Research Council of Canada.

[14] http://ddmal.music.mcgill.ca/neon

8. REFERENCES

[1] Ahn, L. V., B. Maurer, C. McMillen, D. Abraham, and M. Blum. 2008. reCAPTCHA: Human-based character recognition via web security measures. *Science* 321 (5895): 1465–8.

[2] Barton, L. W. G. 2002. The NEUMES project: Digital transcription of Medieval chant manuscripts. In *Proceedings of the International Conference on WEB Delivering of Music*, 211–8.

[3] Catholic Church. 1963. *The Liber Usualis, with introduction and rubrics in English*. Tournai, Belgium: Desclée.

[4] Doan, A., R. Ramakrishnan, and A. Y. Halevy. 2011. Crowdsourcing systems on the world-wide web. *Communications of the ACM* 54 (4): 86–96.

[5] Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley.

[6] Hankinson, A., P. Roland, and I. Fujinaga. 2011. The Music Encoding Initiative as a document-encoding framework. In *Proceedings of the International Conference on Music Information Retrieval*, Miami, FL, 293–8.

[7] Morent, S. 2001. Representing a Medieval repertory and its sources: The music of Hildegard von Bingen. *Computing in Musicology* 12: 19–31.

[8] Newby, G. B., and C. Franks. 2003. Distributed proofreading. In *Proceedings of the Joint Conference on Digital Libraries*. 361–3.

[9] Paulson, L. D. 2005. Building rich web applications with AJAX. *IEEE Computer* 38 (10): 14–7.

[10] Pugin, L. 2009. Editing Renaissance music: The Aruspix project. In *Digitale Edition zwischen Experiment und Standardisierung Musik - Text - Codierung*, 147–56.

[11] Roland, P. 2002. The Music Encoding Initiative (MEI). In *Proceedings of the International Conference on Musical Applications Using XML*, 55–9.

[12] Vigliensoni, G., J. A. Burgoyne, A. Hankinson, and I. Fujinaga. 2011. Automatic pitch detection in printed square notation. In *Proceedings of the International Society for Music Information Retrieval Conference*, Miami, FL, 423–8.