Dencil S. Wilmot
6.UAP Final Report
Professor Eran Egozy
21 May 2018

Automating Graphical Feedback for ConcertCue.com's Human Controller

Background

Concert Cue

The Problem

Classical music is still actively being performed, but without a mandatory classical education, and with such a wide range of styles and composers to choose from, it is infeasible for the average concert goer to be familiar with all of the pieces that can be performed. Currently, they have no choice but to familiarize themselves with the roster as best they can in advance, or enjoy the performance without preparation. This hinders the concert experience because the audience members may miss exciting, yet subtle intricacies of more complicated pieces simply due to lack of preparation.

The Solution

Concertcue.com is a web app that displays, on your smartphone, discrete pieces of musical information, aptly-named "cues", to concert goers in real time. Cues can be as specific as "Listen to the modulation here" or "notice the viola soli here" or more general, only providing historical information about the composer or the piece (See Figure 2). While still in its early stages, Concertcue.com hopes to enrich the average concert goer's experience by providing valuable auditory/historical information about all of the compositions in a performance roster at a given concert in real time.

Existing Framework

Concertcue.com is currently hosted by a private server which interacts with a controller server and a user's mobile device as a concert is being performed. The controller handles measure advancement while the server updates the site to present the user different auditory or historical cues (as diagrammed in figure 1). Currently, the controller server is completely manually operated: a human has to, while looking at a score of the concert and listening to the performance, advance the measure number to send cues to the user. This continuous glancing at both the printed score and the screen is difficult enough to cause confusion and leads to mistakes.

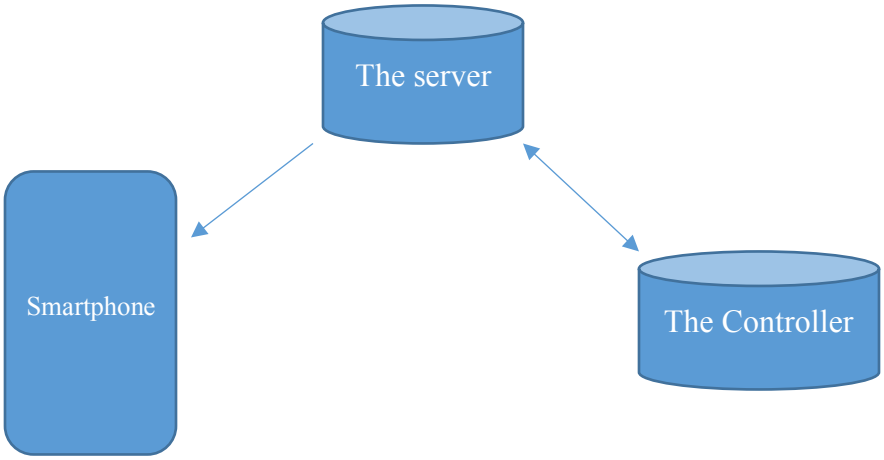

Figure 1- Diagram of the current concertcue.com framework. My project will be done primarily in the realm of the controller.

Existing, Relevant Technology

The two primary goals of this project concern processing/digitizing music and the real-time display of musical information. We will now explore existing technology around these two fields.

Processing/Digitizing Print Scores

As music and technology have evolved over the years, there has been an increasing desire to digitally store and interact with music. One of the most popular ways to do this is by interacting with a digital copy of the score. With the dawn of widely-accessible music annotation software like Musescore 2.0 (2009), Noteflight (2012) and proprietary software like Sibelius (2001), we can now interact with these digital scores more fluidly, thanks to real-time playback and instrument sound synthesis. Many of these software store these digital copies of scores as MusicXML files, the universal standard for representing Western musical notation digitally (Good 2001). Many of these software are also distributed with additional software that enables

the translation of print scores (saved as pdfs or even a collection of image files) to this XML-based file format.

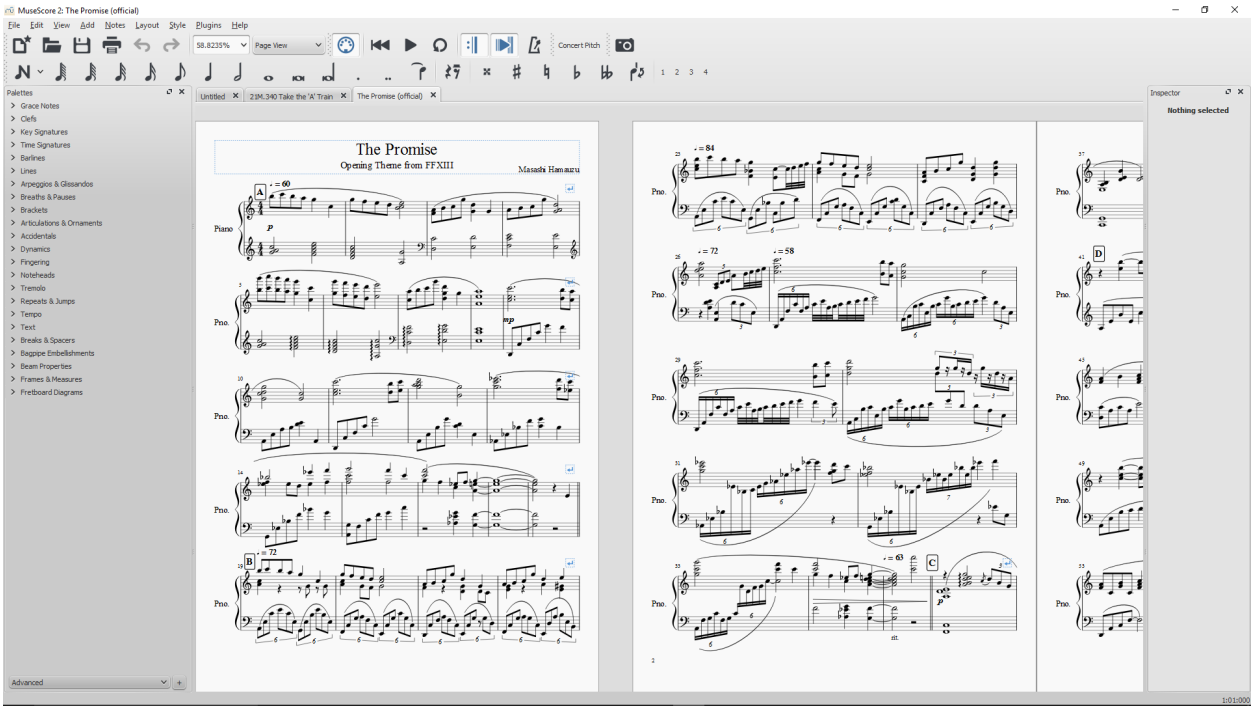

Figure 2- Musescore 2, a common music annotation software.

Another common way to store/interact with music is by digitally recreating the sounds of the score and manipulating these sound files. The most common file formats for managing music in this way are MIDI (1996), WAV, and Flacc. Files are represented, visually, as digital audio signals and can be transformed with software like Audacity, Sonic Visualizer, or Garage Band. This approach de-emphasizes the original score to make music manipulation/interaction more accessible to people without training in western classical music notation. Many software packages exist to convert print scores directly to MIDI files and most of them use some variation of optical music recognition which, as a process, will be discussed in more detail in a future section.

These two formats of digitization, one that enhances the score and one that renounces it completely in favor of a digital audio signal, constitute almost all of the ways that technology has enabled people to interact with music (excluding music in video games, which is outside of the scope of this project). Of all of the formats we now have to represent music, digitally or physically, each format has its own distinctive features and provides different advantages to manipulation. While generally comprehensive, these output formats are not immediately conducive to improving Concert Cue.

Real-Time Display of Musical Information

Having some graphical feedback for music while it's playing is one of the many innovations made possible by more advanced technology. Almost all of the music annotation software packages previously mentioned have both some way of creating sound that corresponds to the

notes/instruments in the score and displaying which sounds in the score were being played (by highlighting measures or individual notes). When music is instead translated into a digital audio signal, having a lined display or scrolling view that changes as a function of time creates a similar visual distinction between the parts of the music being played and the parts that aren't. These digital audio signals can also be separated by frequency composition (to isolate different instruments) and aligned in a grid to further add meaning to the playback experience. Together, these methods provide a sufficient amount of graphical feedback in music to enable musicians and non-musicians to visually engage with the material.

Optical Music Recognition (OMR) and Optical Character Recognition (OCR):

Optical Character Recognition (OCR) scans an image and tries to create a symbolic representation of its contents, usually by parsing/identifying one character/glyph at a time. The advantage of Optical Music Recognition (OMR) as a related field is that produced symbolic representation is intended to have musical significance. This makes the conversion from print scores to a more abstract representation like MusicXML files reasonable. Often, OMR software can output images or PDFs of the print scores into one of the more popular filetypes previously mentioned (musicXML or MIDI). There are many open-source and proprietary software that can perform OMR on sheets and are often packaged with music annotation software directly.

Project Description

The core goal of this project is to automate the graphical feedback for ConcertCue's human controller to reduce the difficulty of advancing the cues in real time. More specifically, by integrating score information onto the same screen as the cues, we can eliminate the need to hold or reference a print score at any point during a performance and, therefore, centralize the act of advancing the cues to the controller's primary field of vision: the computer monitor. As project milestones were completed and implementation/design decisions were made, the scope of the project was refined to ensure completion by the end of the term. This project can be broken into two major components, which are discussed in more detail in the following sections:

1. An offline method for processing sheet music to collect graphical information about the locations of barlines.
2. A real time display for the information from step 1 (designed in a way that will facilitate integration into concertcue.com)

Offline Processing of Sheet Music

The minimum functionality of any real time display for Concert Cue's human controller is measure highlighting. The first step to highlighting a measure is, given sheet music or a digitized score, identifying the locations of all of the measures. Because measures in western classical notation begin and end with barlines, the task for this section becomes twofold:

1. Using OMR software to identify the graphical locations of the barlines. Because barlines are vertical line segments, this task requires finding the pixel location of the start and end of these line segments relative to the rest of the score.
2. Finding some convenient way to store this barline information for future real-time display.

Choosing an OMR Software

The decision to choose Audiveris (Bitteur 2014), an open-source OMR software, over any other OMR software (proprietary or otherwise) was affected by multiple factors like ease-of-use (how easy the software was to install and use), developer support (is the software still being updated/patched), effectiveness (how good the software is at identifying barlines), and core functionality (how easy is it to save the barline information to an unconventional file format).

Of the two open-source projects, OpenOMR and Audiveris, Audiveris seemed like a spiritual successor to OpenOMR because the latter hadn't been updated in upwards of 5 years and the two projects shared similar functionality and effectiveness. Of the proprietary software, a few options originally stood out as contenders, namely, Visiv's SharpEye and Capella-scan. The latter was only available on windows and packaged with an existing music annotation software, which limited scalability enough to stop considering it. Visiv's SharpEye was both more scalable, by supporting all major operating systems, and more reputable because of the use of its SDK in other proprietary OMR software. However, Visiv's SharpEye only supported saving to Midi and MusicXML filetypes, leaving little to no possibility of isolating any of the barline's pixel coordinates before it was saved to these file types. Other cons like limited documentation and little control over the effectiveness of the software were irrelevant if the information couldn't be isolated.

Even though Audiveris also did not clearly support this access of internal representations, because Audiveris is an open-source project, it was very possible to make changes to the codebase to add the functionality necessary for the project. With all of this in mind, Audiveris was the only OMR software that wouldn't require interacting with information stored in MusicXML or Midi files. In addition, the neural net used by Audiveris could be re-trained to improve effectiveness if necessary, which could justify the time required to learn the codebase. Other pros like comprehensive documentation, active developer support, and no financial cost secure Audiveris as the best OMR software for the project.

Isolating Barline Data

Because the project demanded isolating graphical locations of the barlines and Audiveris didn't originally support saving score information to a file type that was conducive to accessing that barline information, a critical element of this project was a modification to Audiveris' source code that records the barline information in a separate JSON file as the software is processing the score.

Why use JSON?

Javascript Object Notation (JSON) is a data-interchange format that is both easy for humans to read and reason about and easy for computers to parse and to edit (Crockford 2006). Because Audiveris is written in Java and the eventual real-time display of the information will be in Javascript (two languages that support reading/parsing JSON files), a well-organized JSON file would be an easy way to interface between them. In addition, a JSON file would allow me to impose meaningful structure on the barline data, minimizing the amount of processing that needs to be done in real-time later. ECMAScript 6 (Engelschall 2016) has well-supported libraries like jQuery to interpret/write JSON strings while Java has multiple versions of JSONSimple (Yidong 2014).

Audiveris' Data Storage

Audiveris defines a hierarchy of abstract data types, almost all of which have inherent meaning in western classical notation (e.x. page, system, measure, barlines etc.), to represent the different symbols/functions in western classical music notation. Then, over a 20-step process, Audiveris continuously updates the mapping between marking on the input score and the abstract data types. We will examine all of these steps and their role in the OMR processing in the next section. Generally, by finding and accessing the method that updates Audiveris' internal representation of the score, we successfully managed to record all instances of potential barlines as Audiveris detects them.

Audiveris' OMR Steps:

A brief overview of the major functions of each of the 20 Audiveris processing steps and their relevance to the project goal of identifying barline information. While steps have to be done in order, the user is free to temporarily stop processing and save the state of the software after any single step. Because barlines are necessarily vertical line segments, many of the steps will have no effect on barline identification.

Step 1: LOAD

At this step, Audiveris collects all of the input files and separates them into greyscale "sheets" and, if the native UI is also running, loads them onto the screen. An Audiveris "sheet" is approximately 1 complete page of a print score (excluding edge cases near the ends of some symphonic movements). This step also ensures that individual pages of the score are prepared for parallelization in subsequent steps, if specified. This step begins when a file is selected and does not modify barline data.

Step 2: BINARY

At this step, Audiveris converts the greyscale images into black-and-white images so that the sheets will be ready for processing. This step is important for barline information, because stray vertical marks or faded sections of the score are either ignored, if the contrast is too low, or accentuated. While these changes then influence the effectiveness of the barline identification in later steps, there isn't specifically any barline identification in this step.

Step 3: SCALE

At this step, Audiveris scans the sheet to try and find the mean distance between the staff lines. This will be used to create a symbolic notion of spacing or "scale" for the sheet. This step standardizes the differences in density of musical information per unit area across different scores. This step is also not directly relevant to identifying barlines.

Step 4: GRID

This step is where most of the barline identification occurs because, in this step, Audiveris only scans the sheets for staff lines, barlines, separate systems, and different parts. These parts of a sheet are mostly structural and are often represented by sections of line segments. While the data collected here can be refined slightly by the "MEASURE" step, the accuracy gained is minimal. Therefore, the current implementation of the project only uses barline data collected at this step.

Step 5: HEADERS

This step scans the sheet for key signatures, time signatures, and clefs and is therefore not at all useful for barline identification.

Step 6: STEM_SEEDS

This step scans the sheet for any vertical line segments which are component pieces of note stems. The stem of a note is the vertical line that attaches to the note's head and, if applicable, the note's flag. This step is not useful in barline identification.

Step 7: BEAMS

This step scans the sheet looking for markings that can be interpreted as note beams. Beams are the horizontal lines that can connect notes with duration less than one-quarter beat. This step is not useful in barline identification.

Step 8: LEDGERS

This step scans the sheet for horizontal line segments that can be interpreted as ledger lines. Ledger lines are primarily used to notate pitches above or below the standard staff. This step is not useful in barline identification.

Step 9: HEADS

This step scans the sheet for circular markings that can be interpreted as note heads or whole notes. This step is not useful in barline identification.

Step 10: STEMS

This step just builds connections between existing "heads" and "beams" without actually scanning the score. This step is not useful in barline identification.

Step 11: REDUCTION

This step analyzes all of the collected symbolic information gathered so far to narrow down the possible interpretations of notes in the current sheet. If for any reason, Audiveris had previously indicated that two marks on the sheet couldn't possibly exist at the same time and preserve the musical validity of the sheet, this is the step where it chooses the more likely interpretation. This step is not useful in barline identification.

Step 12: CUE_BEAMS

This step scans the sheet for smaller beams that could be interpreted as cues for another part. In Western classical notation, sometimes composers indicate what one instrument is playing on another instrument's part (usually while the second instrument is resting). These additional indicators are not meant to be played and are called "cues." This step is not useful in barline identification.

Step 13: TEXTS

This step scans the sheet for any text and calls an internal OCR software to interpret it. This step is not useful in barline identification.

Step 14: MEASURES

This step analyzes all of the existing barline data to symbolically define the measures in a given sheet. Because measures are a collection of at least two barlines and other musical objects, the goal of the project could have been to retrieve measure information instead of barline information. However, because barlines are symbolically simpler objects than measures, Audiveris has a higher identification accuracy with barlines. To take advantage of this higher accuracy, this project ignores the effects of measure interpretation on barline data.

Step 15: CHORDS

This step analyzes existing note heads to define "chords." Chords are a collection of notes that are played at the same time so this step is performed without scanning the sheet and therefore is not useful in barline identification.

Step 16: CURVES

This step scans the sheet for curved markings that could be interpreted as slurs, accents, or endings. This step is not useful in barline identification.

Step 17: SYMBOLS

This step scans the sheet looking for previously unmarked symbols with a fixed shape. Objects like rests, accidentals, flags, dots, and tuples are indistinguishable from other objects of the same type, which makes them easiest to interpret after all of the musical objects that vary between scores have been identified. Because barlines are properties of the score that aren't of fixed shape, this step is not useful in barline identification.

Step 18: LINKS

This step analyzes all of symbol information collected in the previous steps and performs a reduction (like in step 11) to eliminate impossible combinations of symbols in the sheet. For example, because it is impossible to have three "sharp" signs modifying the same note, some of these interpreted marks must be incorrect. This step is not useful in barline identification.

Step 19: RHYTHMS

This step primarily analyzes the measure and symbol information collected so far to try and identify/interpret rhythmic patterns within measures. This step is not useful in barline identification.

Step 20: PAGE

The final step of Audiveris, this step is responsible for grouping all of the individual sheets analyzed thusfar into higher-order structures called "books" and for grouping individual systems in a sheet into one "page." Because this step does very little analysis and is mostly organizational, this step is not useful in barline identification.

JSON Format

The JSON file that holds the barline pixel information is formatted hierarchically to reflect some of the structure of the original score (see Figure 3 for an example). The two keys "sheetNumber" and "systemNumber" correspond to the unique sheet Id and system number of the most recently

scanned page and is used only for partial verification that all of the expected barlines were recorded. To understand the rest of the structure, we will theoretically construct it from the smallest data structure to the largest. Each endpoint of every barline has an x-coordinate and a y-coordinate that define their location relative to the top left corner of the score. Arrays of x-y "endpoints" are combined into arrays that represent completed barlines, because every barline has exactly two endpoints. These arrays of barlines are then indexed first by the "sheetNumber" and then by the "systemNumber" of their corresponding sheet using JSON objects.

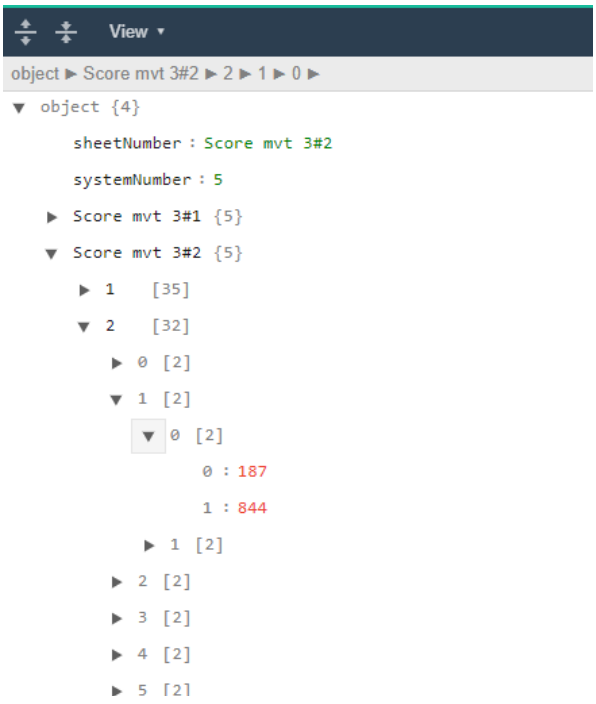

Figure 3 An example, organized view of a valid JSON file

Real-Time Display

The next step after acquiring barline data, stored as a separate JSON file, is to create an interface for Concert Cue's human controller to clearly distinguish measures on a digital score (see Figure 4).

The interface features an image of one of the pages of the score on the left side and controls for highlighting the next measure or selecting a different page on the bottom. The right hand side is reserved for meta-controls like filepath selection. As of right now the arrow buttons for measures

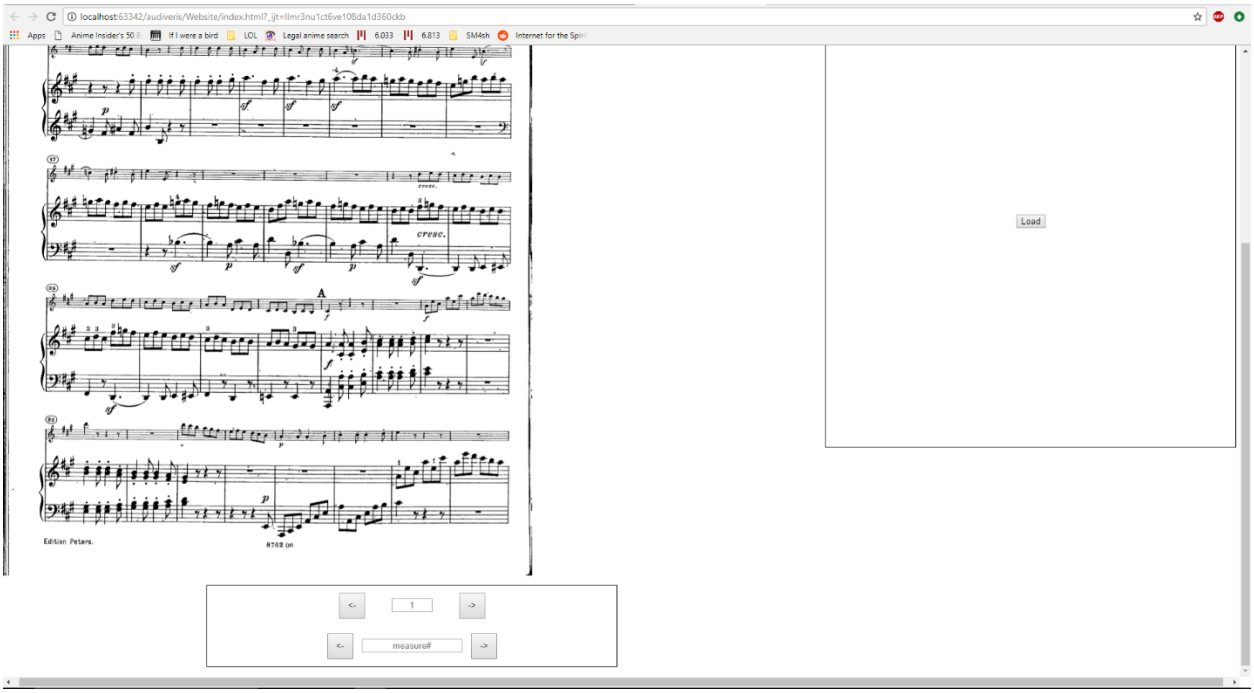

Figure 4: Current Implementation of the Website

and score pages works as intended but the input boxes are disabled (and only display what measure/page is currently being displayed).

Optimally, as the human operator is listening to the performance and advancing the Controller's "current measure", that measure will be highlighted to indicate the temporal position in the digital score, focusing the operator's attention on the screen.

Next Steps/Future Projects

Optimized JSON recording

Currently, my modification of Audiveris loads and saves the JSON every time a new barline is detected so that, in the case of a crash, as much barline data is preserved. However, the associated performance tradeoff is not worth that much fault tolerance because any crashes in Audiveris will always be sufficient cause to restart the software. In addition, because there was no attempt at modifying the Audiveris' original UI to streamline this data collection for the project, future work can be done to edit command-line arguments or the native UI to enable/disable this additional functionality.

Modularity of the Real-Time Display

The current implementation of the display serves more as a proof-of-concept than as a fully-functioning JS library or web plugin, which limits its immediate application to Concert Cue or other contexts. Future work can be done to make the real-time display more modular, including asynchronous file loading with jQuery and a fully functional API.

Conclusion

I have made significant progress to automate graphical feedback for Concert Cue's human controller by using Audiveris, an open-source OMR software, to isolate barline location data from sheet music or digitized scores and then displaying that information on a website, where the controller can view/highlight different measures on a digital copy of the score.

References

[Audacity 2018] Team, A. (2012). Audacity (Version 2.0. 2). Audio editor and recorder.

[Bitteur 2014] Bitteur, H. (2014). Audiveris.. https://github.com/Audiveris/audiveris

[Capella 2018] Capella Software (2018). Scan sheet music with capella-scan. Sold at https://www.capella-software.com/us/index.cfm/products/capella-scan/info-capella-scan/

[Crockford 2006] Crockford, D. (2006). The application/json media type for javascript object notation (json).

[Engelschall 2016] Engelschall, R. S. (2016). ECMAScript 6—New Features: Overview & Comparison. URL https://babeljs. io/docs/plugins.

[Good 2001] Good, M. (2001, December). MusicXML: An internet-friendly format for sheet music. In XML Conference and Expo (pp. 03-04). Republished at https://pdfs.semanticscholar.org/5617/972 667ff794da79a4cbb6b985e85f8487ddd.pdf

[MIDI 1996] The Complete MIDI 1.0 Detailed Specification. Document version 96.1. Los Angeles: The MIDI Manufacturers Association (1996).

[Mori 1999] Mori, S., Nishida, H., & Yamada, H. (1999). Optical character recognition. John Wiley & Sons, Inc..

[Musescore 2009] Bonte, T. (2009). MuseScore: Open source music notation and composition software. Technical report, Free and Open source Software Developers' European Meeting, 2009. http://www. slideshare. net/thomasbonte/musescore-at-fosdem-2009.

[Noteflight 2012] Noteflight, L. L. C. (2012). What is Noteflight. http://www. noteflight. com/info/faq.

[Sibelius 2001] Finn, B., & Finn, J. (2001). Sibelius: The music notation software. http://www.avid.com/sibelius-ultimate

[Visiv 2008] Visiv Ltd (2008) Visiv SharpEye Music Scanning http://www.visiv.co.uk/

[Yidong 2014] Fang Yidong (2014). Json-simple. https://code.google.com/archive/p/json-simple/