Interoperable Digital Musicology Research via `music21` Web Applications

Michael Scott Cuthbert, Beth Hadley, Lars Johnson, Christopher Reyes
Department of Music and Theater Arts
Massachusetts Institute of Technology
Cambridge, Mass., USA

{cuthbert, bhadley, larsj, chrisrr}@mit.edu

Abstract

Digital humanities practices applied to musical scores have the potential to open up vast new datasets and avenues for research in musicology and are beginning to transform the field of musical research. Yet beyond the common difficulties of all digital humanities projects, significant problems arise in digital musicology that are unique to the structure of musical scores and the lack of available tools for manipulating scores. Performing analysis tasks often requires specialized tools that have high barriers to entry, such as compiling, choosing a particular operating system, and converting data between divergent formats. The “webapps” module of the open-source `music21` toolkit provides the architecture to connect various digital musicology projects. It makes standard but time-consuming musicological tools available to less technologically sophisticated users while providing tremendously varied developmental options to technically-inclined researchers. The authors propose a JSON format for encoding both score data and manipulations to/analysis of scores that can easily be used by backend systems besides `music21`, whether specialized for musical analysis or for other digital humanities and machine learning tasks. The article ends by stressing, with examples, the continued need for standalone musical analysis systems even in a world of easily available web systems.

Keywords: Digital Musicology, scores, music web systems

1. Background: Digital Musicology today

Musicological research, particularly of western classical music, has long relied on the intense study of small numbers of individual works looking for particularly distinctive, inspiring, or unusual moments in single scores. Comparative research among scores or repertories has been out of favor since the middle of the century (Cook, 2004) because of inaccuracies (particularly a bias towards western, often Germanic, forms) and an inability to cope systematically with large corpora. Computational approaches to repertories have been embraced in the past twenty years by several projects, but they have not been the norm in musicology due to the difficulty in obtaining computer-encoded versions of scores and in particular the absence of easy-to-use software packages for examining, analyzing, and manipulating these scores.

`Music21` (Cuthbert & Ariza, 2010) is an open-source object-oriented toolkit built in Python for digital and computational musicology. The toolkit builds on the strengths of earlier applications, such as the Humdrum toolkit (Huron, 1997), but adds to it an object-oriented framework that allows users to find desired data more quickly and easily. First released in 2008 for all standard operating systems (including Windows, OS X, and Unix variants), the `music21` toolkit is now in its fourteenth release and the first non-beta version was released on June 14, 2012. The rapid adoption of `music21` for use by computational musicology projects has made it close to a new de facto standard for computer-aided work, but difficulties in increasing its use among less technically minded musicologists has necessitated recent work in building web applications to take advantage of its power while making it simpler to use and eliminating the need for installation.

2. The Present and Future Need for Web Applications in Digital Musicology

Over the past fifteen years, web applications have dominated the field of computational musicology tools by providing musicologists with immediate access to music datasets and simple analytical tools. However, without providing an infrastructure for customization, research is commonly limited to the materials provided through the site, leaving little room for creative development and investigation.

We take the project Kernscores (Sapp, 2008) as exemplifying both the great potential and binding limitations of current musicological web application systems. Like most digital musicology sites, it uses URL-encoded commands accessed via websites to transform data into a variety of musical formats and give the results of simple analytical processes such as key analysis or piano roll diagrams of the pieces. These analyses have great potential, yet the currently available methods come with significant drawbacks. The most obvious is that the tools can only be applied to the scores made available by the developers—a problem shared with nearly all similar sites. These scores need to be encoded in formats that are either not in general use (e.g., Humdrum/Kern) or cannot represent standard notational symbols that are important to researchers and performers (e.g., MIDI which stores the notes D♯ and E♭ as the same pitch and cannot encode tempo markings such as allegro moderato). More significantly for developers, the URL-encodings are not documented and the code for the backend systems are generally not released, making it impossible for outside developers to expand the system.

Although `music21` has been designed to be easy enough for a professional musicologist without previous programming experience to learn to use in a few weeks, even this requirement presents too high of a bar for many users. Web applications offering even simple commands that process user-uploaded data and return results designed for users to view or hear without further computational processing can be incredibly valuable to researchers of all technical backgrounds.

Additionally, a service-oriented architecture (SOA) allows more advanced web developers to easily integrate complex computational methods into their own web applications. Web applications are currently being produced for many platforms, and the easy integration of computational back-end tools would make such applications even more powerful.

Finally, computer scientists working on improving generalized algorithms for classification of data are another untapped audience needing web applications for musical scores. A researcher wishing to see if her algorithm for clustering data can also work on musical scores will seldom have time or expertise to learn a specialized system for feature extraction of musical data; she and her team will be searching for already created sets of feature data (such as the Million Song Dataset gives for audio data (Bertin-Mahieux, et. al, 2011)) or a way of easily obtaining these features from data gathered from other sources. A service-oriented architecture is critical for the needs of researchers only tangentially connected to digital musicology. Such a web architecture would allow this researcher to leave specialized feature extraction tasks to musicological experts and focus on her own expertise in algorithmic design.

3. `Music21` Web Applications

Since its conception, `music21` has provided a modular infrastructure for manipulating and analyzing scores. This makes it ideal for providing the link between accessible web environments and sophisticated music research. Beginning with the 1.0 release, `music21` includes a module designed for developing a service-oriented architecture utilizing the full suite of analysis tools provided by `music21`. The webapps SOA eliminates many hurdles to utilizing the `music21` toolkit by placing it in a web-based setting, yet still provides users and developers unparalleled freedom.

`Music21` web applications import and export data in a variety of formats, catering to a wide range of user communities. Computer-aided musicology has always depended on utilizing various data formats to encapsulate the vast variety of information extracted from music queries. For example, `music21` web applications export textual and numeric data in formats ranging from simple text or JSON, to .csv and spreadsheet formats, to graphical plots. It supports numerous music notation formats, including MusicXML and Lilypond as well as MIDI and even Braille translation. Additionally, these web applications can take advantage of being embedded in modern web browsers by enabling live, editable notational output through the Noteflight (Berkovitz, 2008) Flash-based plugin and manipulable high- quality Canvas and SVG graphics through the open-source VexFlow (Cheppudira, 2010) JavaScript library. Users can run web applications using the 10,000 scores in the `music21` corpus or assemble their own corpora. Providing such versatility to users ensures a broad compatibility with other music-based websites and independent stand-alone music applications.

`Music21`'s implementation of the VexFlow JavaScript library is particularly important for future adoption of web applications for musical scores. Prior to the creation of VexFlow, no freely available way of rendering musical data on the Internet as a viewable score was feasible. Previous attempts such as the Mediawiki extension to Lilypond (www.mediawiki.org/wiki/Extension:LilyPond) posed serious security hazards and required translating existing MIDI, MusicXML, and other score files into a new format. With `music21`'s adoption of VexFlow and the SOA, any Internet user can render a data file in one of numerous formats as a score for viewing within a web page or other JavaScript/HTML5-compatible application. Future work on this module will add JavaScript callbacks from the VexFlow code to the `music21` SOA enabling interactive musical markup, annotation, and editing.

4. Example Uses of `Music21` Web Applications

The `music21` service-oriented architecture can be used for a variety of purposes. Applications can be developed in which a simple click of a button can trigger advanced analysis routines. For example, commands easily automated via `music21` webapps include output of range and key data, detection of contrapuntal anomalies such as parallel and direct fifths, transformation of a collection of pieces to the same key or meter, and various feature extraction methods. One commonly used method of `music21` is the "chordify" command which takes in an entire score, measure range, or collection of parts, and reduces it to a series of chords representing the music sounding at each moment in the score. This reduced score is much easier to understand at a quick glance than a full score. The tremendous modularity innate in `music21` methods and objects allows identification and analysis of music scores not possible via static interfaces similar to previous musicology sites where both user input and analysis tools are limited.

For the advanced user, the `music21` service-oriented architecture may be used as a platform upon which more complex web applications may be built. An example demonstrating the versatility of the webapp architecture coupled with the interoperability offered by the toolkit is a tool we created for analyzing a student's music theory assignment for contrapuntal writing errors (See Figure 1). Using the `music21` webapp architecture, the student's assignment passes easily from third-party notation software to analysis methods within the toolkit that identify areas of concern in the work. The tool then returns a pre-graded score, either to the student or the professor, along with text describing each error. Of particular interest to educators is the automatic identification of violations of common-practice rules of counterpoint, such as motion by parallel fifth or dissonant harmonic intervals. In developing this app, we extended and customized the existing `music21` methods of analysis, creating specialized `music21` objects to encapsulate individual elements within the score, such as linear segments, vertical slices of simultaneously sounding objects, and two by two matrices of notes. Elements identified as errors were colored, and text output further explained the algorithm's observation (such as between which notes the parallel fifths exist, or the name of the dissonant interval). This data is packaged into a JSON data structure and provided directly to the client (either a web browser or the open-source MuseScore notation software (Brontë, et. al., 2008) completing the service to the user. This service-oriented architecture for music is under consideration to become the backbone for music courses in the developing MITx/EdX open educational platform.

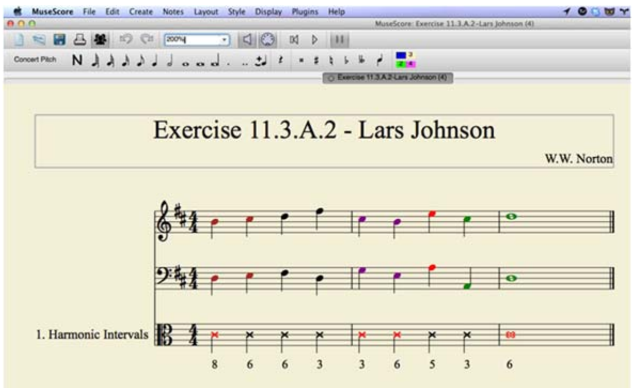

Figure 1: Screenshot displaying the use of this webapp embedded as a plugin for the open-source notation software MuseScore used as part of an automatic "pre-grading" system for music theory teaching. A full video showing this demonstration is available at http://www.youtube.com/watch?v=5VBfag3YwIs .

5. Service-Oriented Architecture in `music21`: the webapps library

To enable development of interoperable webapps utilizing the full suite of computational tools, the `music21` toolkit includes an extensive service-oriented architecture. It consists of Python classes and functions used to parse a server request, execute the desired commands, and return content to the user in an appropriate format. The flexible nature of the architecture allows it to use a single URL to handle any requests to the server wishing to use `music21`. These requests can come from a variety of sources, including HTML form POSTs, AJAX requests, or even web requests from a plugin in an open source notation application. The commands used by the requests can either be commands built in to `music21` or custom commands created by the user.

The core of the module involves two objects: an Agenda, and a CommandProcessor. An Agenda object is a dictionary-like structure that specifies data input, requested commands, and a desired output format. A CommandProcessor object takes an Agenda, parses the data input into a format compatible with `music21`, safely executes the commands, and generates the output.

These objects are used in a server application compliant with the Python WSGI interface, a portion of which is shown below. This application can be enabled on an Apache/modWSGI server by adding a few lines to the httpd.conf, as Figure 2 demonstrates.

```
from music21 import *
agda = webapps.makeAgendaFromRequest(requestInput,environ)
processor = webapps.CommandProcessor(agda)
processor.executeCommands()
(responseData, responseContentType) = processor.getOutput()
```

Figure 2: Code for setting up a `music21` web application.

The code shown is representative of the steps involved in processing a request. First, the POST data and GET data from the request are combined into an Agenda object. The post data can be url-encoded form data, multipart form data, or a JSON string. In this way a single mount point can be used to serve a variety of request types.

Figure 3 shows an example of the typical JSON formatted input to the webapp interface. This text encodes commands to use `music21` parse a Bach chorale from the corpus, transpose that chorale by a perfect fifth, then return the chordified score in VexFlow format. Should the user wish to view their score in a different `music21`-supported output format, such as MusicXML, Braille, Lilypond, or MIDI, only a one-word change to this JSON format is necessary.

```
{ "dataDict": { "workName": { "data": "'bwv7.7'" } },
  "commandList": [
    { "function": "corpus.parse",
      "argList": [ "workName" ],
      "resultVar": "chorale" },
    { "caller": "chorale",
      "method": "transpose",
      "argList": [ "'p5'" ],
      "resultVar": "choraleTransposed" },
    { "caller": "choraleTransposed",
      "method": "chordify",
      "resultVar": "choraleChordified" }
  ],
  "outputTemplate": "templates.vexflow",
  "outputArgList": ["choraleChordified"]
}
```

Figure 3: An example JSON request to return a Bach chorale (BWV 7 movement 7) as a chordal reduction, transposed up a perfect fifth as a VexFlow Canvas.

If an appName is specified in one of the request fields, additional data and commands are added to the agenda. This flexibility allows for the creation of applications in which the majority of the commands are specified by the server and only a subset of the data is specified by the user for each request. For instance, by specifying a "featureExtractorApp," as the appName, each request would only need to include the name of the feature they would like to extract and the zipfile containing the scores, without explicitly needing to specify the individual commands necessary for feature extraction and machine learning of musical data (Cuthbert, Ariza & Friedland, 2011).

The command processor then takes the agenda and parses its input data into primitives or `music21` objects. Although most of the values arising from POST and GET fields start as type string, the processor will determine if the string was intending to be a number, boolean, list, etc. and save its value accordingly. Additionally, `music21` is compatible with a wide variety of symbolic music formats (MusicXML, Humdrum/Kern, abc, MIDI, etc.) and can convert fields of those types into corresponding `music21` objects.

Next, the command processor executes the commands specified by the agenda. To avoid the security risk of executing arbitrary code while still maintaining the flexibility of the architecture, the server checks that each requested command is allowed to be executed on the server and only interacts with a set of variable bindings internal to the processor.

Finally, the processor generates the output of the results. The elements of the Agenda specify the output format which can be of a wide variety of types, including an html page with a score displayed in an SVG or Flash embed, a downloadable MusicXML or comma-separated value file containing analysis results, or simply the raw JSON of selected variables that can be decoded using JavaScript in a client HTML page.

A video demonstrating this system is viewable at http://ciconia.mit.edu/feature-extraction.wmv and the software itself is at
http://ciconia.mit.edu/music21/featureapp/uploadForm
Examples of sample webapps are available at http://ciconia.mit.edu/music21/webapps/client/.

6. Cloud Computing and Web Services

Repertorial analysis requiring the best analytical methods might run hundreds of times per score on a corpus of tens of thousands of scores. The `music21` service-oriented architecture provides the infrastructure necessary to command complex and computationally intensive analysis. However, such tasks might take hours to run and provide little to no real-time feedback during processing. Thus, it has become apparent that integrating more powerful processing power would make `music21` webapp services even more accessible. Our recent work has included research into providing cloud computing functionality to `music21` analysis routines via Amazon Web Services and the Python map-reduce module, mrjob (Yelp, 2009)

Any webapp routine that can be abstracted into multiple independent tasks benefits greatly from the additional computing power provided through cloud computing. Processing time can be greatly decreased by implementing a standard MapReduce algorithm (Dean & Ghemawat, 2004) to distribute processing of hundreds or thousands of files over a network of independent computers. The Python library mrjob accesses Amazon Web Services and can be utilized to prepare MapReduce algorithms employing `music21` analytical methods. Due to the modularity of the `music21` service-oriented architecture, webapps can be developed to provide quicker access to `music21` processes via the Amazon Cloud. These webapps would route input data from the user, such as a corpus of music files, establish an SSH connection with EC2 instances provided by Amazon, deploy the job specified, and wait while the data is processed. The resulting output would be passed back to the web interface and displayed to the user in a fraction of the time it would take the user to run the same analysis algorithm on a local computer. After implementing this process in a test run examining bass motion over thousands of popular music leadsheets we recorded promising improvements in the time taken in processing many scores.

By adding the component of cloud computing to our already existing service-oriented `music21` architecture, the limit of computational power and time is tremendously alleviated. Integrating cloud computing into a pre-existing web service allows musicologists great freedom in both developing and running research studies.

7. Limitations of Web-systems and the Co-existence of Stand-alone systems in Digital Musicology

While web-based applications will open up many new avenues for research and data exchange, downloadable applications to be run on individual users' systems will need to continue to be developed. To start, unless a system is implemented entirely in JavaScript, users' queries need to be parsed and understood by a traditionally based backend system. As long as such an engine exists, there is little to be gained by limiting programmers' access to this backend, and continued development of server-based systems demand tests that can be executed outside the web system. More complex queries that nest the filtering of musical objects and annotations are much more easily created with short scripts that have direct access to the musical objects. For instance, the research question "does Mozart cadence on first-inversion triads more often on strong beats vs. weak beats in his sonatas written earlier in his life?" is easily answered in `music21` by writing a short module using nested "if," "break," and "getElementsByClass()" statements. A similar web query would be so complex that the designing the command would be a more difficult process than installing the system and writing a script by hand. A researcher must carefully evaluate the advantages to developing a web-based application versus stand-alone scripts, depending on their individual goals, technical background, and time constraints. In addition, while HTML5 simplifies many programming tasks and moves them from the server to the client, it does not contain support for microphone or MIDI input without external plugins (usually Adobe Flash). Thus for many realtime audio and musical applications, standalone versions of the software are needed.

Security and privacy concerns are two other factors to consider when evaluating whether to develop a web-based platform or stand-alone application. Complex queries may require access to the file system or generate huge temporary files, both of which can introduce security holes. Users may not want to trust their private research data to be uploaded to a web server not under their control. This desire may seem paranoid when the only data are musical scores, but `music21` can also correlate score data with physiological response data from listeners and reported musical preferences, all of which could be used to deanonymize survey data. Thus, both security and privacy concerns promote continued development of stand-alone applications.

8. Conclusion and Future Work

Fundamentally, the goal of the `music21` service-oriented architecture is to provide researchers from a wide range of technical backgrounds and disciplines access to powerful musical analysis tools conveniently, efficiently, and quickly. Future development includes expanding the webapp infrastructure to implement a larger suite of customizable `music21` features along with improved computational power via the Amazon Cloud. Modules within the toolkit that require extensive external dependencies, such as "Gregorio" the LaTeX chant notation software, can be adapted to use the SOA to render the notation on a properly equipped external server. Work in the near future will also include extensions to our VexFlow web architecture to enable interactive annotation and editing of SVG-rendered musical scores. The possibilities of service-oriented architectures in computational musicology toolkits such as `music21` are only beginning to be tapped. In the near future music web applications will be among the most important contributors to the exciting cross-disciplinary advancements emerging in digital humanities.

9. Acknowledgements

The development of `music21` and its web applications has been funded by a NEH Digging into Data Challenge Grant as part of the ELVIS project and by the Seaver Institute. Additional support has been provided by the School of Humanities, Arts, and Social Sciences at MIT and support for this presentation has been provided by the Germany Seed Fund of the MIT International Science and Technology Initiatives.

10. References

Berkovitz, J. (2008, continuing). Noteflight: http://www.noteflight.com/

Bertin-Mahieux, T., Ellis, D.P.W., Whitman, B., Lamere, P. (2011). "The Million Song Dataset," *Proceedings of the International Symposium on Music Information Retrieval* 12.

Brontë, T., Froment, N., Schweer, W. (2008, continuing). MuseScore: http://www.musescore.org/

Cheppudira, M.M. (2010, continuing). VexFlow: http://www.vexflow.com/

Cook, N. (2004). "Computational and Comparative Musicology," in *Empirical Musicology: Aims, Methods, Prospects*. Oxford: Oxford University Press, pp. 103–26: 103.

Cuthbert, M.S., Ariza, C. (2010), "`music21`: A Toolkit for Computer-Aided Musicology and Symbolic Music Data," *Proceedings of the International Symposium on Music Information Retrieval* 11, pp. 637–42.

Cuthbert, M.S., Ariza, C., Friedland, L. (2011), "Feature Extraction and Machine Learning on Symbolic Music using the `music21` Toolkit," *Proceedings of the International Symposium on Music Information Retrieval* 12, pp. 387–92.

Dean, J., Ghemawat, S. (2004). "MapReduce: Simplified data processing on large clusters," *Proceedings of the 6th Symposium on Operating System Design and Implementation*, pp. 137–50.

Huron, D. (1997). "Humdrum and Kern: Selective Feature Encoding." In *Beyond MIDI: the Handbook of Musical Codes*. E. Selfridge-Field, ed. Cambridge, Mass.: MIT Press, pp. 375-401.

Sapp, C.S. (2008, continuing). Kernscores. http://kern.ccarh.org/

Yelp (Software company) (2009, continuing). mrjob. http://packages.python.org/mrjob/index.html