VEROVIO: A LIBRARY FOR ENGRAVING MEI MUSIC NOTATION INTO SVG

Laurent Pugin
Swiss RISM Office
laurent.pugin@rism-ch.org

Rodolfo Zitellini
Swiss RISM Office
rodolfo.zitellini@rism-ch.org

Perry Roland
University of Virginia
pdr4h@virginia.edu

ABSTRACT

Rendering symbolic music notation is a common component of many MIR applications, and many tools are available for this task. There is, however, a need for a tool that can natively render the Music Encoding Initiative (MEI) notation encodings that are increasingly used in music research projects. In this paper, we present Verovio, a library and toolkit for rendering MEI. A significant advantage of Verovio is that it implements MEI's structure internally, making it the best suited solution for rendering features that make MEI unique. Verovio is designed as a fast, portable, lightweight tool written in pure standard C++ with no dependencies on third-party frameworks or libraries. It can be used as a command-line rendering tool, as a library, or it can be compiled to JavaScript using the Emscripten LLVM-to-JavaScript compiler. This last option is particularly interesting because it provides a complete in-browser music MEI typesetter. The SVG output from Verovio is organized in such a way that the MEI structure is preserved as much as possible. Since every graphic in SVG is an XML element that is easily addressable, Verovio is particularly well-suited for interactive applications, especially in web browsers. Verovio is available under the GPL open-source license.

1. INTRODUCTION

A few decades ago, rendering music notation by computer almost exclusively targeting printed output, most often in Postscript of PDF formats. Today, partly in response to the development of MIR applications, rendering of music notation can be necessary in a wide range of contexts, for example within standalone desktop applications, in server-side web application scenarios, or directly in a web browser. For example, music notation might need to be rendered for displaying search results or for visualizing analysis outputs. Another example is score-following applications, where the passage currently played needs to be displayed and possibly highlighted. Rendering music notation by computer, however, is a complex task. Powerful music notation rendering engines exist in commercial and open-source notation editors, but these are usually not very modular and cannot easily be integrated within other applications. Other rendering engines, such as LilyPond [13] or Mup [1], can be used; however, they usually require the encoding to be converted to a particular typesetting input syntax. Their architectures and dependencies also often limit the contexts in which their use is possible.

In recent years, the Music Encoding Initiative (MEI) has been increasingly adopted for music research projects [6]. Its large scope (MEI can be used to encode a wide range of music notations, from medieval neumes to common Western music notation), modularity, rich metadata header and numerous other features, including alignment with audio files or performance annotations, make it appropriate for a wide range of MIR applications. Unfortunately, most of the solutions currently available for rendering MEI involve a conversion to another format, either explicitly or internally in the software application used for rendering.

In this paper, we present the Verovio project, a library and toolkit for rendering MEI natively in SVG. Its purpose is to provide a self-contained typesetting engine that is capable of creating high-quality graphical output and that can also be used in different application contexts. In the following section, we describe previous work and existing solutions for rendering MEI and the use of SVG for music notation. We then introduce Verovio, describe the MEI structure on which it is built, outline its programming architecture, and highlight features currently available. We then present possible uses and output examples and conclude the paper with the future work that is planned for Verovio.

2. PREVIOUS WORK

One currently available option for rendering MEI is conversion to another format in order to use existing tools that do not support MEI. For software applications or rendering engines that support the import of the MusicXML interchange format, MEI can be converted with the mei2musicxml XSL stylesheet [9]. Another option is to convert MEI directly to a typesetting format, such as Mup. Mup is a C rendering engine that was made open-source in 2012. It uses its own typesetting syntax and produces high quality Postscript output. The conversion of MEI to Mup can be achieved in one step using the mei2mup XSL stylesheet [8]. A similar approach is possible for rendering MEI in a web browser, using a conversion to the ABC format. ABC is an encoding format primarily targeting material with fairly limited notational features, such as folk and traditional melodies. It can be rendered in a web browser with the abcjs renderer [15], and the conversion from MEI to ABC can be achieved with the mei2abc converter [5]. There is also a new JavaScript library, MEItoVexflow [18], that makes it possible to render MEI directly in web browsers using the Vexflow API [12]. Another tool for rendition of MEI online is Neon.js [3]. The tool not only renders, but also acts as a full online editor for neumatic medieval notation.

SVG for music notation has been used in several projects. One early attempt was made in 2003 for converting MusicXML to SVG using XSLT [14]. A framework with an editor was also developed for outputting SVG from GuidoXML notation as part of a dissertation thesis [2]. With MEI, SVG rendering was used for the first time in the DiMusEd project, a critical edition of songs of Hildegard von Bingen (1098-1179) [11]. In this web-based edition of neumatic notation, MEI rendering is performed on the server side with a custom rendering engine. There are also attempts to use XSLT to transform MEI to SVG directly in the browser. This approach is used in mono:di, the transcription software of the Corpus Monodicum editorial project sponsored by the Akademie der Wissenschaften und der Literatur in Mainz, also focused on medieval notation [4]. Finally, SVG is a possible back-end for the aforementioned Vexflow API in conjunction with the Raphael JavaScript library.

These solutions all have strengths and drawbacks in terms of compatibility, usability, speed, output quality, and music notation features available. Many of them, however, have limitations when the format to which MEI is converted for rendering does not support some features encoded in the MEI source or has a different structure, with the consequence that part of the encoding will be lost in conversion, or not rendered appropriately.

3. VEROVIO

3.1 MEI structure

The MEI schema provides multiple options for structuring the musical content. The most widely-used option is the score-based structure, where all the parts of a musical score are encoded together in the same XML sub-tree. The MEI schema also includes a part-based option, where each part is stored in a separate XML sub-tree. The choice between these options can depend not only on the type of document being encoded but also on the type of application. The Verovio library was designed as a direct implementation of the MEI structure. However, since it is rendering-focused, it is built around another content organization of MEI, a page-based customization more appropriate for graphical display. In a rendering task, the page (or more generically, the rendering surface) is a required high-level entity on which elements can be laid out by the rendering process. The page-based customization is a more fitting alternative data organization that provides a page top-level entity. It prioritizes the hierarchy that is treated as secondary when encoded with milestone elements `<pb>` in other MEI representations.

In the page-based customization, the content of the music is encoded in `<page>` elements that are themselves contained in a `<pages>` element within `<mdiv>` as shown in Figure 1. A `<page>` element contains `<system>` elements. From then on, the encoding is identical to standard MEI. That is, a `<system>` element will contain `<measure>` elements or `<staff>` elements that are both un-customized, depending on whether or not the music is measured or un-measured, respectively. Since the modifications introduced by the customization are very limited, the Verovio library can be used to render un-customized MEI files. When loading un-customized MEI documents, some MEI elements are loaded by Verovio and converted to a page-based representation. Typically, `<pb>` milestone elements are converted to `<page>` container elements. Conversely, `<section>` container elements are converted to `<secb>` milestone elements.

```xml
<body>
  <mdiv>
    <pages>
      <page page.width="2108" page.height="2970" page.leftmar="20">
        <system system.leftmar="50" system.rightmar="50">
          <scoreDef>
            <staffGrp symbol="line">
              <staffDef n="1" clef.line="2" clef.shape="G" />
            </staffGrp>
          </scoreDef>
          <measure n="1">
            <staff n="1">
              <layer n="1">
                <!-- ... -->
              </layer>
            </staff>
          </measure>
        </system>
      </page>
    </pages>
  </mdiv>
</body>
```

Figure 1. The page-based MEI structure used by Verovio. The `<mdiv>` element contains `<pages>`, `<page>` and `<system>` elements.

3.1.1 Layout and positioning

In addition to making rendering simpler and faster, the idea of the page-based customization is also to make it possible to encode the positioning of elements directly in the content tree without having to refer to the facsimile sub-tree. The latter traditional approach remains available with the page-based customization for more detailed and more complex referencing to facsimile images. However, the page-based customization introduces a lightweight positioning and referencing system that can be useful when the encoding represents a single source with one image per page. This is typically the case with optical music recognition applications for which the encoding of the position of each encoded element is necessary. Another possible use is the creation of overlay images to be displayed on top of facsimile images where the position of each symbol also needs to be encoded. Verovio supports both positioned elements and automatic layout. Automatic layout will be executed when un-customized MEI files are rendered.

3.1.2 Additional supported formats

In addition to MEI, Verovio can render Plain and Easy (PAE) code [7] and DARMS code [16]. PAE and DARMS encodings are widely used for encoding incipits, including those for the Répertoire International des Sources Musicales (RISM) project. In Verovio, these formats are converted to MEI internally, which means that the toolkit can also be used to convert them to MEI for purposes other than rendering.

3.2 SVG output

One significant advantage of SVG rendering over other formats (e.g., Postscript or PDF) is that it is rendered natively in most modern web browsers with no plug-in required. Because SVG is XML, it has an advantage over raster image formats that every graphical element is addressable, making it well-suited for interactive applications. In a web environment, this makes it easy to highlight notes or symbols, for example. In addition, since SVG is a vector format, the output can also be used for high-quality printing.

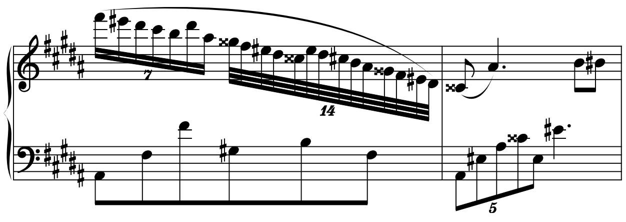
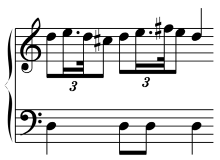


Figure 2. The output of Vervovio for two bars. The built-in layout engine of Verovio avoids symbol collisions as much as possible.

One interesting feature of Verovio is that the SVG is organized in such a way that the MEI structure is preserved as much as possible. For example, a `<note>` element with an `@xml:id` attribute in the MEI file will have a corresponding `<g>` element in the SVG with an `@class` attribute equal to `"note"` and an `@id` attribute corresponding to the `@xml:id` of the MEI note. This makes interaction with the SVG very easy. The hierarchy of the elements is also preserved. For example, in MEI, a `<beam>` can be the child element of a `<tuplet>`, but the opposite is also possible. The hierarchy is fully preserved in the SVG as shown in Figure 3.

```
<tuplet xml:id="t1" num="3" numbase="2">
  <beam xml:id="b1">
    <note xml:id="n1" pname="d" oct="5" dur="8" />
    <note xml:id="n2" pname="e" oct="5" dur="16" dots="1"/>
    <note xml:id="n3" pname="d" oct="5" dur="32" />
    <note xml:id="n4" pname="c" oct="5" dur="8" accid="s"/>
  </beam>
</tuplet>
<beam xml:id="b2">
  <tuplet xml:id="t2" num="3" numbase="2">
    <note xml:id="n5" pname="d" oct="5" dur="8" />
    <note xml:id="n6" pname="e" oct="5" dur="16" dots="1"/>
    <note xml:id="n7" pname="f" oct="5" dur="32" accid="s"/>
    <note xml:id="n8" pname="e" oct="5" dur="8"/>
  </tuplet>
</beam>
```



```
<g class="tuplet" id="t1" >
  <g class="beam" id="b1" >
    <g class="note" id="n1" >...</g>
    <g class="note" id="n2" >...</g>
    <g class="note" id="n3" >...</g>
    <g class="note" id="n4" >...</g>
  </g>
</g>
<g class="beam" id="b2" >
  <g class="tuplet" id="t2" >
    <g class="note" id="n5" >...</g>
    <g class="note" id="n6" >...</g>
    <g class="note" id="n7" >...</g>
    <g class="note" id="n8" >...</g>
  </g>
</g>
```

Figure 3. Comparison of MEI and SVG file structures. The hierarchy of the MEI is preserved in the SVG.

3.3 Programming architecture

Verovio is designed as a fast, portable, lightweight tool usable as a one-step conversion program. It is written in pure standard C++ with no dependencies on third-party frameworks and libraries. This ensures maximum portability of the codebase. Verovio implements its own rendering engine, which can produce SVG with all the musical symbols embedded in it. The musical glyphs are themselves SVG graphics that are included in the Verovio output. This means that no external font needs to be included in the SVG generated from Verovio, limiting dependencies and reducing as far as possible any potential compatibility issues between SVG rendering engines.

The Verovio rendering engine itself is defined as an abstract class, and the SVG output is the default concrete class. This makes it relatively easy to implement a rendering back-end different from SVG (e.g., PDF, or HTML Canvas), if necessary.

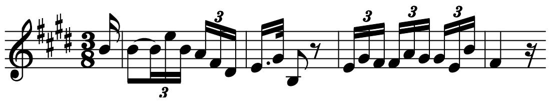
The Verovio toolkit has several options for controlling the output. These include options for defining the page size (i.e., the surface, or `<svg>` element size), for setting the amount of zoom, and for choosing whether layout information contained in the MEI file must be taken into account. When there is no layout information provided in the MEI file (no system or page breaks, for example), or when the option for ignoring them is selected, Verovio will extrapolate the necessary layout information.

3.4 Features

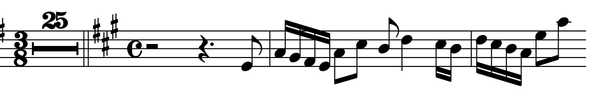
Verovio currently supports the basic features of simple common Western music notation and mensural notation.

Table 1 shows a list of music notation snippets rendered with Verovio. Figure 4 illustrates how the SVG output of Verovio can be used as facsimile overlay when the positioning feature of the MEI page-based customization is used. The example also illustrates mensural music notation support.

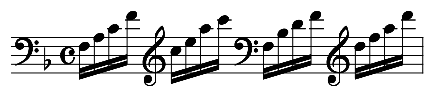
Beams and tuplets



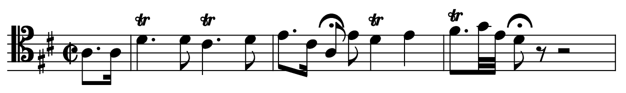
Measure rests and key and time signature changes



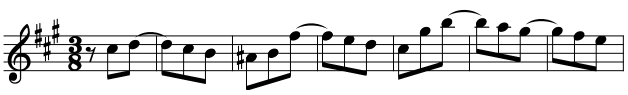
Clef changes



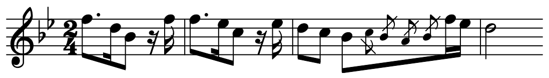
Trills and fermata



Ties



Grace notes (accaciature)



Grace notes (appogiature)

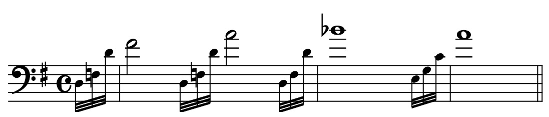


Table 1. A list of music notation snippets rendered with the Verovio toolkit. The basic features of simple common Western music notation are accounted for.

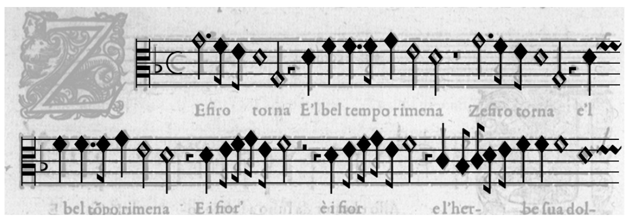

Efiro torna E'l bel tempo rimena Zefiro torna e'l
bel tẽpo rimena E i fior' e i fior e l'her- be ſua dol-

Figure 4. An example of the output of Verovio placed back on top of a facsimile image and acting as transcription overlay. In this case, positioning information was available in the page-based MEI encoding.

4. USE OF VEROVIO

4.1 C++ tools and library

Several use cases can be imagined for the Verovio toolkit. First of all, it can be built and used as a standalone command-line tool. This option is well-suited to scripting environments and applications. The command-line tool can be used to render music notation files (in MEI, PAE or DARMS) into SVG. It can also be used to convert DARMS or PAE to MEI. Another option is to use Verovio as a music notation rendering library that can be statically or dynamically linked to full applications. In such cases, it is also relatively easy to implement another drawing back-end for the corresponding C++ graphic environment for the music to be rendered directly on the screen. This is the case with the Aruspix optical music recognition software application where Verovio provides a screen rendering using a wxWidgets back-end instead of the standard SVG one. This approach is conceivable for any C++ graphical environments, be they cross-platform, like the Qt or JUCE toolkits, or platform specific.

4.2 JavaScript toolkit

The Verovio toolkit can also be compiled to JavaScript using the Emscripten LLVM-to-JavaScript compiler [19]. In this case, it behaves similarly to the command-line tool but in the web browser context. This approach is particularly interesting because it provides a complete in-browser music MEI typesetter that can be easily integrated into web-based applications.

Emscripten does not directly translate C++ into JavaScript. Instead it takes the LLVM (Low Level Virtual Machine) byte code generated by the Clang compiler from the C++ code as a base for the conversion to JavaScript. This has several advantages. Most importantly, the level of completeness in terms of C++ language feature support is extremely high since the idiomatic features of C++ did not have to be explicitly translated into JavaScript in the Emscripten compiler (only the translation from LLVM was necessary). In fact, for the Verovio toolkit, the Emscripten compiler is applied on exactly the same codebase as the C++ compiler, and no change to the code had to be done for this to work. Only the compilation makefile is different.

Another advantage of this approach is that the JavaScript produced is very fast because it benefits from all the code optimization performed by the Clang compiler when generating the LLVM byte code. Furthermore, in addition to standard JavaScript, Emscripten can also generate asm.js code, a subset of JavaScript that has the advantage of being highly optimizable. On web browsers that support asm.js (currently Firefox, Chrome and Opera), the execution speed is only up to about 1.6 times slower than with the native C++ executable. Table 2 shows the system time required to load an MEI file of 120 pages of music (7 MB) and for displaying the first page with the native executable and with three web browsers. The figures are the median value of the operation repeated 100 times.

	Native	Firefox	Chrome	Safari
System time in sec.	0.657	1.054	1.364	1.811
Comparison to native	-	1.6	2.1	2.8

Table 2. The system time in seconds for loading an MEI files (120 pages, 7 MB) and for displaying the first page. The second line gives the ratio with the native executable time for the three web browsers used for comparison.

The JavaScript version of the Verovio toolkit is easy to use in web environments. It is packed in one single file which size is only about 1.2 MB. It is available as a JavaScript class, and all the options of the command-line version are supported in the toolkit. The options can be passed to the toolkit in JSON format, and the SVG output can be directly fed to HTML objects for display. The Figure 5 shows a HTML and Javascript code snippet for loading an MEI file using a jQuery HTTP GET request.

```
<script src="verovio-toolkit.js"></script>
<!-- The div where we are going to insert the SVG -->
<div id="output"/>
<script type="text/javascript">
  /* Create the Vevorio toolkit instance */
  var vrvToolkit = new verovio.toolkit();
  /* Load the file using HTTP GET */
  $.get( "mei-file.mei", function( data ) {
    /* Render the data */
    var svg = vrvToolkit.renderData(
                data + "\n",
                JSON.stringify({ inputFormat: 'mei' }) );
    /* Insert the data as content of the #output div */
    $("#output").html(svg);
  });
</script>
```

Figure 5. A JavaScript example for loading an MEI file. The toolkit parameters can be set using JSON.

The layout of the MEI data is performed on loading. Once the file is loaded into memory, it remains accessible in the toolkit instance. The class provides methods for getting the number of pages or for navigating through them, making it convenient to integrate the toolkit in a JavaScript application.

The Figure 6 shows a screenshot of a web application where the toolkit was turned into an online MEI file viewer. The application works on desktop computers but also on tablets and mobile devices. The JavaScript toolkit has been tested with recent versions of the most widely used web-browsers. Internet Explorer requires at least version 10.

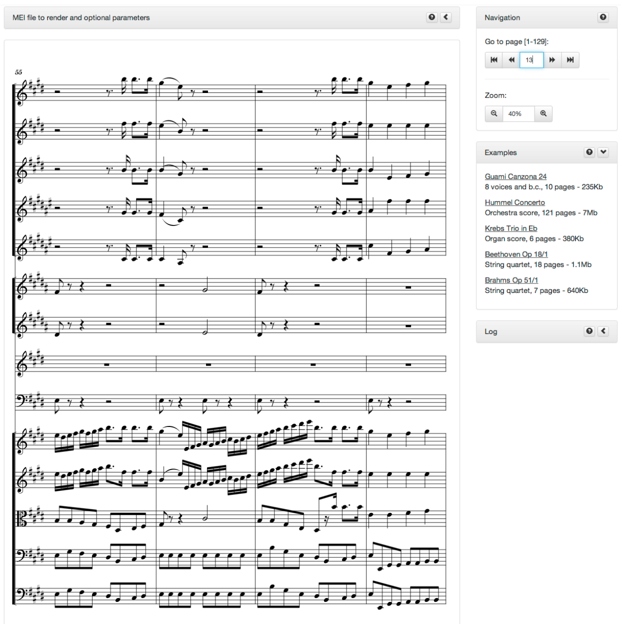

MEI file to render and optional parameters
55
Navigation
Go to page [1-129]:
13
Zoom:
40%
Examples
Guami Canzona 24
8 voices and b.c., 10 pages - 235Kb
Hummel Concerto
Orchestra score, 121 pages - 7Mb
Krebs Trio in Eb
Organ score, 6 pages - 380Kb
Beethoven Op 18/1
String quartet, 18 pages - 1.1Mb
Brahms Op 51/1
String quartet, 7 pages - 640Kb
Log

Figure 6. An example of a web-based MEI viewer built with the Verovio toolkit. Large MEI files can be loaded and displayed in the web browser in a very convenient way.

5. CONCLUSION AND FUTURE WORK

Verovio is a toolkit for rendering MEI in SVG that can be used in different application environments, including online. It is designed with MEI in mind, making it the right basis for implementing encoding features that are specific to MEI. It will avoid problematic situations that occur when using rendering engines based on other formats and that implement a different data structure. Even if at this stage, the supported features can be in some cases more limited than with other rendering options, Verovio already implements many important features for rendering both common Western music notation and mensural notation.

Current work on Verovio includes the adoption of the Standard Music Font Layout (SMuFL) [17] for supporting other fonts converted to SVG glyphs, the improvement of the SVG structure and adding support for additional MEI elements and attributes. The priority is given to features specific to MEI. The future work will include the development of a prototype for making Verovio a possible basis for an online MEI editor. It will also include the creation of an MEI application profile for Verovio using the TEI One Document Does-it-all (ODD) approach. The corresponding XSL stylesheets for converting to it other MEI profiles will also be provided. Adding the import of other encoding formats is also envisaged in the future.

6. AVAILABILITY

Verovio can be downloaded from http://www.verovio.org and is available under the GPLv3 open-source license. The website also includes documentation on currently available features.

7. REFERENCES

[1] Arkkra Enterprises, *Mup.* <http://www.arkkra.com>

[2] G. A. Bays: *ScoreSVG: A New Software Framework for Capturing the Semantic Meaning and Graphical Representation of Musical Scores Using Java2D, XML, and SVG.* Diss. Georgia State Univ., 2005.

[3] G. Burlet, A. Porter, A. Hankinson, and I. Fujinaga: "Neon.js: Neume Editor Online," *Proceedings of the 13th International Society on Music Information Retrieval Conference,* pp. 121–6, 2012.

[4] Corpus Monodicum, *mono:di.* <http://monodi.corpus-monodicum.de>

[5] Edirom, *mei2abc*. <https://github.com/edirom/mei2abc>

[6] A. Hankinson, P. Roland, and I. Fujinaga: "The Music Encoding Initiative as a document-encoding framework," *Proceedings of the 12th International Society on Music Information Retrieval Conference,* pp. 293–8, 2011.

[7] J. Howard: "Plaine and Easie code: A code for music bibliography," in Selfridge-Field, E. (Ed.), *Beyond MIDI: The Handbook of Musical Codes*. The MIT Press, Cambridge, pp. 362–72, 1997.

[8] MEI, *mei2mup*. <http://code.google.com/p/music-encoding/source/browse/trunk/tools/mei2mup>

[9] MEI, *mei2musicxml*. <https://code.google.com/p/music-encoding/source/browse/trunk/tools/mei2musicxml>

[10] MEI-incubator, *page-based customization*, <https://code.google.com/p/mei-incubator/source/browse/page-based>

[11] S. Morent: "Digitale Edition älterer Musik am Beispiel des Projekts TüBingen," in *Digitale Edition zwischen Experiment und Standardisierung. Musik – Text – Codierung*, pp. 89–109, 2009.

[12] M. Muthanna, *VexFlow.* <https://github.com/0xfe/vexflow>

[13] H. W. Nienhuys and J. Nieuwenhuizen: "LilyPond, a system for automated music engraving," *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, pp. 167–72, 2003.

[14] L. O'Shea: "Stirring XML: Visualizations in SVG: MusicML2SVG," *Proceedings of the SVGOpen2003 Conference,* pp. 2–6, 2003.

[15] P. Rosen, *abcjs*. <http://github.com/paulrosen/abcjs>

[16] E. Selfridge-Field: "DARMS, its dialects, its uses," in Selfridge-Field, E. (Ed.), *Beyond MIDI: The Handbook of Musical Codes*. The MIT Press, Cambridge, pp. 163–74, 1997.

[17] Steinberg, *Standard Music Font Layout.* <http://www.smufl.org>

[18] TEI Music SIG, *MEItoVexFlow*. <http://github.com/tei-music-sig/meitovexflow>

[19] A. Zakai: "Emscripten: an LLVM-to-JavaScript compiler," *Companion to the 26th Annual ACM OOPSLA Conference*, pp. 301–12, 2011.